IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: James CRAWFORD	Art Unit	: 2141
Serial No.	: 09/597,784	Examiner	: April Baugh
Filed	: June 19, 2000		
Title	: DIRECT FILE TRANSFER BETWEEN SUBSCRIBERS OF A COMMUNICATIONS SYSTEM		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. §1.131

I, James Crawford, hereby declare as follows:

1.   I have read and understood the text of U.S. Application No. 09/597,784 (the '784 application), which discloses an invention for which I am the inventor.

2.   On or prior to April 27, 2000, I reduced to practice the methods, computer programs, apparatus, and user interface described in paragraphs 4 and 5 of this document.

3.   The attached pages are photocopies of:

a) a screen shot of a directory containing the AIM installer program (Exhibit 1) showing a date of March 1, 2000 associated with the program. The AIM installer program is used to install the Windows AIM version 3.5.1856 binary ("the AIM program").

b) a redacted source code listing of the portion of the AIM program used for file transfer functionality (Exhibit 2). Each line of code is numbered.

c) a redacted source code listing for the portion of the AIM program used to setup connections between the client and the host (Exhibit 3)

d) a screen shot of a user interface of the AIM program showing a file transfer window that enables a user to set file transfer preferences (Exhibit 4).

e) a screen shot of a user interface of the AIM program showing an instant messaging interface depicting a subscriber having a user identity "oscarlogan" selecting an option to get a file from another subscriber having a user identity "OscaRaina" (Exhibit 5).

Applicant : James CRAWFORD
Serial No. : 09/572,953
Filed : May 18, 2000
Page : 2 of 5

Attorney's Docket No.: 06975-
053001 / Communications 06

f) a screen shot of a user interface of the AIM program showing a list of files that the user identity "oscarlogan" may attempt to get from the user identity "OscaRaina" (Exhibit 6).

g) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" after selecting a file from the list of files displayed in Exhibit 6 (Exhibit 7).

h) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" indicating the status of the file transfer (Exhibit 8).

i) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" indicating a request from the user identity "OscaRaina" to get files from the disk directory belonging to the user identity "oscarlogan" (Exhibit 9).

4.   With respect to independent claims 1, 14, 29-31, and 36 of the '784 application, I implemented and practiced a method, a computer program and an apparatus that transferred one or more files between clients.

Specifically, the following was implemented and practiced as evidenced by the source code listings of Exhibits 2 and 3:

(a) a connection was established with a communications system host

- Exhibit 3 –
    - Lines 2740-2773   SessSignOn
    - Lines 1346-1396   ConnCreate
    - Lines 1514-1577   ConnConnect
    - Lines 491-495   connDoServerLookup
    - Lines 171-208   connLookupHost
    - Lines 1048-1056 connWndProc
    - Lines 599-641   connEventLookupComplete
    - Lines 506-517   connDoServerConnect
    - Lines 248-262   connConnectToHost
    - Lines 1063-1080 connWndProc
    - Lines 2268-2306 connEventRecvReady

Applicant : James CRAWFORD
Serial No. : 09/572,953
Filed : May 18, 2000
Page : 3 of 5

Attorney's Docket No.: 06975-
053001 / Communications 06

      o   Lines 2159-2265 connReceiveBlock

      o   Lines 4628-4747 connProcessFLAP

      o   Lines 4052-4060 connProcessSignOn

      o   Lines 739-743   connEventAwaitChallangeComplete

      o   Lines 569-592   connDoValidation

      o   Lines 3731-3873 connSendSignOn

      o   Lines 2394-2410 ConnSendPacket

      o   Lines 2333-2391 connEventSendReady

(b) a request to establish a direct connection was sent to or received from a client also connected to the communications system host.

- Exhibit 2 –

      o   Lines 2031-2032 Process Menu command to Get File

      o   Lines 1733-1767 DoStartGet

      o   Lines 1705-1724 RequestAndListen

(c) when the client permitted establishment of the direct connection, a direct socket connection that bypasses the communications system host was established

- Exhibit 2 –

      o   Lines 3300-3323 SockListen

      o   Lines 3338-3373 SockAcceptReady

      o   Lines 3375-3533 SockRecvReady

(d) and a transfer of one or more files from the client was initiated over the direct socket connection.

- Exhibit 2 -

      o   Lines 3108-3131 File Listing Dialog

      o   Lines 1492-1494 handle IDC_GET button from dialog

      o   Lines 938-980   FTGetListItem

      o   Lines 446-462   FTReInitHdr

      o   Lines 3639-3654 SockSend

      o   Lines 464-481   FtInitHdr

o Lines 3248-3827 SockXXX handle TCP i/o

o Lines 292-1158 FtXXX process bytes received from other client and save to disk

5. With respect to independent claim 45 of the '784 application, I implemented and practiced a user interface that controlled file transfers between clients.

Specifically, the following was implemented and practiced as evidenced by Exhibit 9:

(a) A first graphical user interface element is configured to notify an operator of a second client of a request from a first client to establish a direct connection to the second client. The request is communicated to the second client by a communications system host, and the direct connection bypasses the communications system host.

- Exhibit 9 –
  - o The "Get File request from OscaRaina" window.

(b) A second graphical user interface element configured to enable an operator of the second client to authorize establishment of a direct connection and a file transfer over the direct connection.

- Exhibit 9 -
  - o The "OK" button in the "Get File request from OscaRaina" window

6. The AIM program was produced or written by myself or under my direction on or prior to April 27, 2000, and the date from Exhibit 1 supports this fact. The screen shots of the user interfaces relating to the transfer of files between clients were produced using the AIM program.

7. The AIM program corresponds to the method, computer program, apparatus, and user interface described in paragraphs 4 and 5 of this document.
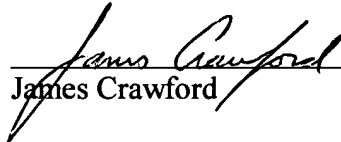
I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are

Applicant : James CRAWFORD
Serial No. : 09/572,953
Filed : May 18, 2000
Page : 5 of 5

Attorney's Docket No.: 06975-
053001 / Communications 06

punishable by fine or imprisonment, or both, under 18 U.S.C. §1001 and that such willful false statements may jeopardize the validity of the application or any patents issued thereon.

Signed and Declared at _Belmont, MA_ this _6th_ day of ~~February~~ August, 2004

James Crawford

declaration over cited art.doc

```
C:\FileTransfer>dir
 Volume in drive C is DRIVE_C
 Volume Serial Number is 0B6F-13EC

 Directory of C:\FileTransfer

01/20/2004  09:37a       <DIR>          .
01/20/2004  09:37a       <DIR>          ..
03/01/2000  05:01a           2,008,104 Windows_AIM_3.5.1856.exe
               1 File(s)        2,008,104 bytes
               2 Dir(s)     985,595,904 bytes free
```

Exhibit 1

```
00292 void FTCloseFileAndSetTime(LPRENDEZVOUSTICKET rTicket)
00293 {
00294     struct _utimbuf times;
00295     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00296     FTHDR *shdr = &lpSrvStruct->sock_hdr;
00297
00298     if (lpSrvStruct->fileP) {
00299         fclose(lpSrvStruct->fileP);
00300         lpSrvStruct->fileP = 0;
00301
00302         times.actime = SWAP4(shdr->dwFiletime);
00303         times.modtime = times.actime;
00304         _utime(lpSrvStruct->dirPath,&times);
00305     }
00306 }
00307
00308 DWORD FTCalcChecksum(FILE* fileP, DWORD fsize)
00309 {
00310     WORD sum = 0;
```

Exhibit 2

```
00311        char buf[4096];
00312        while (fsize) {
00313            int n, nr = (fsize > sizeof(buf)) ? sizeof(buf) : fsize;
00314            n = fread(buf,1,nr,fileP);
00315            if (n != nr) {
00316                sum = 0;
00317                break;
00318            }
00319            sum = ComputeSum(sum, (LPWORD)buf, n);
00320            fsize -= n;
00321        }
00322        fseek(fileP, 0, SEEK_SET);
00323        sum = ~sum;
00324        return (DWORD)(0xffff & sum);
00325 }
00326
00327 DWORD FTRecalcChecksum(DWORD chksum, LPWORD buf, long count)
00328 {
00329        WORD sum = (WORD)(~chksum);
00330        sum = ComputeSum(sum,buf,count);
00331        sum = ~sum;
00332        return (DWORD)(0xffff & sum);
00333 }
00334
00335 BOOL FTIsFileThere(LPSTR path, DWORD ftime, DWORD* chksumP, DWORD* fsizeP)
00336 {
00337        FILE *fileP = fopen(path,"rb");
00338        *fsizeP = 0;
00339        *chksumP = 0;
00340        if (fileP) {
00341            struct _stat fst;
00342            _fstat(fileno(fileP), &fst);
00343            if (fst.st_mtime == (long)ftime) {
00344                *fsizeP = (DWORD)(filelength(fileno(fileP)));
00345                *chksumP = FTCalcChecksum(fileP,*fsizeP);
00346            }
00347            fclose(fileP);
00348            return TRUE;
00349        }
00350        return FALSE;
00351 }
00352
00353 // this insures that there is a \ at the end
00354 void FTSetDirPath(LPRENDEZVOUSTICKET rTicket, LPSTR in, BOOL buddyList)
00355 {
00356        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00357        LPSTR cp, out = lpSrvStruct->dirPath;
00358        if (in)
00359            lstrcpy(out,in);
00360        cp  = _fstrrchr(out,'\\');
00361        if (cp) {
00362            cp++;
00363        } else {
00364            for (cp=out; *cp; cp++) ;
00365            *cp++ = '\\';
00366        }
00367        *cp = '\0';
00368        lpSrvStruct->dirPathOffset = cp;
00369
00370        if (buddyList) {
00371            lstrcpy(cp,rTicket->nickname);
00372            while (*cp)  cp++;
00373            lstrcpy(cp,".lst");
```

```
00374        }
00375 }
00376
00377 BOOL FTConstructDirListing(LPSTR filelib, DWORD *totSizeP, WORD *numFilesP)
00378 {
00379        WIN32_FIND_DATA ffData;
00380        HANDLE ffh;
00381        FILE* fileP;
00382        LPSTR listName = LISTNAME;
00383        LPSTR logName = LOGNAME;
00384        LPSTR cp = filelib;
00385        while (*cp) cp++;
00386
00387        *totSizeP = 0;
00388        *numFilesP = 0;
00389        lstrcpy(cp,listName);
00390        fileP = fopen(filelib,"w");
00391        if (!fileP) {
00392            return FALSE;
00393        }
00394        lstrcpy(cp,"*");
00395        ffh = FindFirstFile(filelib,&ffData);
00396
00397        // TODO: how to tell path is a dir and not a * cmd?  isdir = TRUE;
00398        if (ffh != INVALID_HANDLE_VALUE) {
00399            do {
00400                if (ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
00401                    // char dirpath[MAX_PATH];
00402                    // TODO: go into subdirs also  (recursively call this func??)
00403                } else if (!lstrcmp(ffData.cFileName,listName) ||
00404                        !lstrcmp(ffData.cFileName,logName)) {
00405                } else if (!(ffData.dwFileAttributes & FILE_ATTRIBUTE_HIDDEN)) {
00406                    char oneline[MAX_LIST_LINE];
00407                    FILETIME locFiletime;
00408                    FILETIME *ftimeP = &ffData.ftLastWriteTime;
00409                    SYSTEMTIME systime;
00410                    if (ftimeP->dwLowDateTime == 0 && ftimeP->dwHighDateTime == 0)
00411                        ftimeP = &ffData.ftCreationTime;
00412                    FileTimeToLocalFileTime(ftimeP, &locFiletime);
00413                    FileTimeToSystemTime(&locFiletime, &systime);
00414                    sprintf(oneline,LINEFMT,
00415                            systime.wMonth,systime.wDay,systime.wYear,
00416                            systime.wHour,systime.wMinute,
00417                            ffData.nFileSizeLow, ffData.cFileName);
00418                    fwrite(oneline,lstrlen(oneline),1,fileP);
00419                    *numFilesP += 1;
00420                    *totSizeP += ffData.nFileSizeLow;
00421                }
00422            } while (FindNextFile(ffh,&ffData));
00423            FindClose(ffh);
00424        }
00425        fclose(fileP);
00426        lstrcpy(cp,listName);
00427        return (FTSortFile(filelib,LINEOFF_NAME));
00428 }
00429
00430 BOOL FTNextFile(LPSRVSTRUCT lpSrvStruct, LPSTR path)
00431 {
00432        BOOL ret = TRUE;
00433        if (path) {
00434            lpSrvStruct->ffh = FindFirstFile(path,&lpSrvStruct->ffData);
00435            if (lpSrvStruct->ffh == INVALID_HANDLE_VALUE)
00436                return FALSE;
```

```
00437        } else
00438            ret = FindNextFile(lpSrvStruct->ffh,&lpSrvStruct->ffData);
00439        while (ret &&
00440            (lpSrvStruct->ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
00441            ret = FindNextFile(lpSrvStruct->ffh,&lpSrvStruct->ffData);
00442        return ret;
00443 }
00444
00445
00446 void FTReInitHdr(LPRENDEZVOUSTICKET rTicket)
00447 {
00448        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00449        FTHDR *shdr = &lpSrvStruct->sock_hdr;
00450
00451        lpSrvStruct->sock_starttime = GetTickCount();
00452        lpSrvStruct->sock_numSent = 0;
00453        lpSrvStruct->sock_numTotal = 0;
00454        lpSrvStruct->status_numTodo = 0;
00455        lpSrvStruct->totalNum = 1;
00456        lpSrvStruct->doneNum = 0;
00457        lpSrvStruct->status_lasttime = 0;
00458        lpSrvStruct->status_lastpercnt = 0;
00459        lpSrvStruct->totalSizeOfDoneFiles = 0;
00460        shdr->dwFilesize = 0;
00461        shdr->dwTotalFilesize = 0;
00462 }
00463
00464 void FTInitHdr(LPRENDEZVOUSTICKET rTicket)
00465 {
00466        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00467        FTHDR *shdr = &lpSrvStruct->sock_hdr;
00468
00469        _fmemcpy(&shdr->bMagic[0], OSCAR_FT_MAGIC, sizeof(shdr->bMagic));
00470        shdr->wHdrType = StateToHdrType(lpSrvStruct->state);
00471        _fmemcpy(&shdr->bCookie[0], rTicket->cookie, sizeof(shdr->bCookie));
00472        shdr->wEncryption = 0;
00473        shdr->wCompression = 0;
00474        shdr->wTotalNumParts = SWAP2(1);
00475        shdr->wNumPartsLeft = SWAP2(1);
00476        shdr->dwTotalRessize = 0;
00477        shdr->dwRessize = 0;
00478        shdr->dwRestime = 0;
00479        shdr->dwResChecksum = 0;
00480        lstrcpy(&shdr->bIDstring[0],OSCAR_CLIENT_ID_STRING);
00481        _fmemset(&shdr->bDummy[0], 0, sizeof(shdr->bDummy));
00482        if ((lpSrvStruct->state == StateFileToSend ||
00483            lpSrvStruct->state == StateListToSend) &&
00484            lpSrvStruct->ffh != INVALID_HANDLE_VALUE) {
00485            WORD tmp;
00486            DWORD fsize, chksum;
00487            struct _stat fst;
00488            int hlen, bnamelen;
00489            FTCloseFileAndSetTime(rTicket);
00490            strncpy(&shdr->bName[0],lpSrvStruct->ffData.cFileName,FNSZ);
00491            bnamelen = lstrlen(lpSrvStruct->ffData.cFileName) + 1;
00492            if (bnamelen >= FNSZ) {
00493                shdr->bName[FNSZ-1] = '\0';
00494                bnamelen = FNSZ;
00495            }
00496            hlen = (sizeof(FTHDR) - FNSZ) + bnamelen;
00497            if (hlen < MIN_HDR_SZ)
00498                hlen = MIN_HDR_SZ;
00499            shdr->wHdrLen = SWAP2(hlen);
```

```
00500        lstrcpy(lpSrvStruct->dirPathOffset,lpSrvStruct->ffData.cFileName);
00501        lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"rb");
00502        lpSrvStruct->sock_numSent = 0;
00503        lpSrvStruct->status_numTodo = 0;
00504        _fstat(fileno(lpSrvStruct->fileP), &fst);
00505        fsize = lpSrvStruct->ffData.nFileSizeLow;
00506        chksum = FTCalcChecksum(lpSrvStruct->fileP,fsize);
00507        shdr->dwTotalFilesize = SWAP4(lpSrvStruct->totalSize);
00508        shdr->wTotalNumFiles = SWAP2(lpSrvStruct->totalNum);
00509        tmp = lpSrvStruct->totalNum - lpSrvStruct->doneNum;
00510        shdr->wNumFilesLeft  = SWAP2(tmp);
00511        shdr->dwFilesize = SWAP4(fsize);
00512        shdr->dwFiletime = SWAP4(fst.st_mtime);
00513        shdr->dwChecksum = (chksum);
00514        shdr->dwNumRecvd = 0;
00515        shdr->dwRecvdChecksum = 0;
00516        shdr->bFlags = 0;
00517        if (lpSrvStruct->state == StateListToSend) {
00518            shdr->bListNameOffset = LINEOFF_NAME;
00519            shdr->bListSizeOffset = LINEOFF_SIZE;
00520            if (lpSrvStruct->sorted)
00521                shdr->bFlags |= FLAGS_SORTED;
00522            else
00523                shdr->bFlags &= ~FLAGS_SORTED;
00524        }
00525    }
00526 }
00527
00528 // returns 0 for ok, 1 for err
00529 BOOL FTValidateHdr(LPRENDEZVOUSTICKET rTicket)
00530 {
00531     // insure it is a real header and one we are expecting; we dont want to
00532     // allow hacker client to change headers on us to do something
unauthorized
00533     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00534     FTHDR *shdr = &lpSrvStruct->sock_hdr;
00535     int hdrType = shdr->wHdrType;
00536     int state = lpSrvStruct->state;
00537     if (_fmemcmp(shdr->bMagic,OSCAR_FT_MAGIC,sizeof(shdr->bMagic)) ||
00538         _fmemcmp(shdr->bCookie, rTicket->cookie, sizeof(shdr->bCookie)) ||
00539         SWAP2(shdr->wHdrLen) < MIN_HDR_SZ ||
00540         !(hdrType & SR_MASK) ||
00541         // we shouldnt be recving a header we should be sending
00542         (lpSrvStruct->type & SR_MASK) == (hdrType & SR_MASK)) {
00543         return 1;
00544     }
00545     shdr->bIDstring[IDSZ-1]=0;
00546     shdr->bName[FNSZ-1]=0;
00547     return 0;
00548 }
00549
00550 void ShowStatusWindow(LPRENDEZVOUSTICKET rTicket, int state)
00551 {
00552     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00553     if (ISFT_SERVER(lpSrvStruct) &&
00554         ProfGetLong(PROF_USER, FT_KEY, FT_PUT_NO_STATUS))
00555         ShowWindow(rTicket->hDlgWnd, SW_HIDE);
00556     else
00557         ShowWindow(rTicket->hDlgWnd, state);
00558 }
00559
00560 // returns 0 for ok, 1 for err, 2 for done
00561 BOOL FTProcessHdr(LPRENDEZVOUSTICKET rTicket)
```

```
00562 {
00563      TCHAR buf[512],buf1[512];
00564      LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00565      FTHDR *shdr = &lpSrvStruct->sock_hdr;
00566      int hdrType = shdr->wHdrType;
00567      int state = lpSrvStruct->state;
00568      BOOL sendhdr = -1;
00569
00570      // Note this increments state to the next level
00571      if (hdrType == HDR_TYPE_FILE_TO_SEND) {
00572          if (state == StateFileToSend) {
00573              BOOL normal = TRUE;
00574              DWORD fsize, chksum, fsize1, chksum1;
00575              lpSrvStruct->totalNum = SWAP2(shdr->wTotalNumFiles);
00576              lpSrvStruct->totalSize = SWAP4(shdr->dwTotalFilesize);
00577              if (rTicket->cmdID != CMDID_SEND_FILE) {
00578                  int nd,nf;
00579                  FTSetDirPath(rTicket,0,0);
00580                  nd = (int)(lpSrvStruct->dirPathOffset
00581                          - &lpSrvStruct->dirPath[0]);
00582                  nf = lstrlen(shdr->bName) + 1;
00583                  if (nd + nf > MAX_PATH)
00584                      nf = MAX_PATH - nd;
00585                  strncpy(lpSrvStruct->dirPathOffset,shdr->bName,nf);
00586                  lpSrvStruct->dirPath[MAX_PATH-1] = '\0';
00587                  if (CheckForSecurityHoles(shdr->bName)) {
00588                      SET_RENDEZVOUS_DECLINE(rTicket);
00589                      return 1;
00590                  }
00591              }
00592              FTCloseFileAndSetTime(rTicket);
00593              if (FTIsFileThere(lpSrvStruct->dirPath,SWAP4(shdr->dwFiletime),
00594                          &chksum,&fsize)) {
00595                  UINT id = 0;
00596                  // file already exists; decide if we need to resume it
00597                  fsize1 = SWAP4(shdr->dwFilesize);
00598                  chksum1 = (shdr->dwChecksum);
00599                  if (chksum == 0) {
00600                      // file exists but wrong timestamp or 0 length
00601                      LoadString(lpOCMInfo->hModule,IDSFT_ErrFileAlreadyExists1,
00602                              buf,sizeof buf);
00603                      sprintf(buf1,buf,lpSrvStruct->dirPath);
00604                      if (lpSrvStruct->flags & (FLAGS_YES|FLAGS_NO) ||
00605                          MessageBox(rTicket->hDlgWnd, buf1, 0,
00606                                  MB_OKCANCEL|MB_DEFBUTTON2|MB_ICONEXCLAMATION)
00607                              ==IDOK) {
00608                          remove(lpSrvStruct->dirPath);
00609                      } else {
00610                          shdr->bFlags |= FLAGS_DONTWANT;
00611                          id = IDSFTP_RecvrDoneDontWant;
00612                      }
00613                  } else if (fsize1 == fsize && chksum1 == chksum) {
00614                      // dont bother with this file
00615                      shdr->dwNumRecvd = SWAP4(fsize);
00616                      shdr->dwRecvdChecksum = (chksum);
00617                      shdr->bFlags |= FLAGS_IDENTICAL;
00618                      id = IDSFTP_RecvrDoneNothing;
00619                  } else if (fsize < fsize1) {
00620                      // local file is smaller; offer to resume
00621                      lpSrvStruct->state = StateFileWantToResume;
00622                      shdr->dwNumRecvd = SWAP4(fsize);
00623                      shdr->dwRecvdChecksum = (chksum);
00624                      normal = FALSE;
```
Page 10

```
00625                         }
00626                         if (id != 0) {
00627                             lpSrvStruct->state = StateFileFooter;
00628                             if (rTicket->cmdID == CMDID_GET_LIST)
00629                                 shdr->bFlags |= FLAGS_CONT;
00630                             AppendMsg(rTicket,id,FALSE,ISFT_RCVR(lpSrvStruct),TRUE);
00631                             lpSrvStruct->doneNum++;
00632                             lpSrvStruct->totalSizeOfDoneFiles += fsize1;
00633                             FTInitFileList(rTicket, FALSE);
00634                             SET_RENDEZVOUS_DONE(rTicket);
00635                             normal = FALSE;
00636                         }
00637                         lpSrvStruct->flags &= ~(FLAGS_NO|FLAGS_YES);
00638                     }
00639                     if (normal) {
00640                         lpSrvStruct->state = StateFileOkToSend;
00641                         lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"wb");
00642                         if (!lpSrvStruct->fileP) {
00643                             LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
00644                                       buf,sizeof buf);
00645                             MessageBox(rTicket->hDlgwnd, buf, 0, MB_OK);
00646                             return 1;
00647                         }
00648                     }
00649                     sendhdr = TRUE;
00650                 }
00651         } else if (hdrType == HDR_TYPE_FILE_OK_TO_SEND) {
00652             if (state == StateFileOkToSend) {
00653                 lpSrvStruct->state = StateFileData;
00654                 sendhdr = FALSE;
00655             }
00656         } else if (hdrType == HDR_TYPE_FILE_WANT_TO_RESUME) {
00657             if (state == StateFileOkToSend) {
00658                 DWORD chksum;
00659                 DWORD nrecvd = SWAP4(shdr->dwNumRecvd);
00660                 DWORD chksumr = (shdr->dwRecvdChecksum);
00661                 BOOL good = FALSE;
00662                 FILE* fileP = fopen(lpSrvStruct->dirPath,"rb");
00663                 if (fileP) {
00664                     chksum = FTCalcChecksum(fileP,nrecvd);
00665                     fclose(fileP);
00666                     if (chksum == chksumr) {
00667                         lpSrvStruct->sock_numSent = nrecvd;
00668                         lpSrvStruct->sock_numTotal += lpSrvStruct->sock_numSent;
00669                         good = TRUE;
00670                     }
00671                 }
00672                 if (!good) {
00673                     shdr->dwRecvdChecksum = 0;
00674                     shdr->dwNumRecvd = 0;
00675                     nrecvd = 0;
00676                 }
00677                 lpSrvStruct->state = StateFileToResume;
00678                 sendhdr = TRUE;
00679             }
00680         } else if (hdrType == HDR_TYPE_FILE_TO_RESUME) {
00681             if (state == StateFileToResume) {
00682                 DWORD nrecvd = SWAP4(shdr->dwNumRecvd);
00683                 lpSrvStruct->state = StateFileOkToResume;
00684                 lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,
00685                                       nrecvd ? "ab" : "wb");
00686                 if (!lpSrvStruct->fileP) {
00687                     LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
```

```
00688                             buf,sizeof buf);
00689                     MessageBox(rTicket->hDlgwnd, buf, 0, MB_OK);
00690                     return 1;
00691                 }
00692                 if (nrecvd) {
00693                     lpSrvStruct->sock_numTotal += nrecvd;
00694                 }
00695                 sendhdr = TRUE;
00696             }
00697         } else if (hdrType == HDR_TYPE_FILE_OK_TO_RESUME) {
00698             if (state == StateFileOkToResume) {
00699                 lpSrvStruct->state = StateFileData;
00700                 sendhdr = FALSE;
00701             }
00702         } else if (hdrType == HDR_TYPE_FILE_FOOTER) {
00703             if (state == StateFileFooter ||
00704                 state == StateFileOkToSend) {
00705                 UINT id = IDSFTP_RecvrDone;
00706                 if (shdr->bFlags & FLAGS_DONTWANT)
00707                     id = IDSFTP_RecvrDoneDontwant;
00708                 else if (shdr->bFlags & FLAGS_IDENTICAL)
00709                     id = IDSFTP_RecvrDoneNothing;
00710                 else if (shdr->dwChecksum &&
00711                     shdr->dwChecksum != shdr->dwRecvdChecksum)
00712                     id = IDSFTP_RecvrDoneBadSum;
00713                 AppendMsg(rTicket, id, FALSE, ISFT_RCVR(lpSrvStruct),TRUE);
00714                 lpSrvStruct->doneNum++;
00715                 lpSrvStruct->totalSizeOfDoneFiles += SWAP4(shdr->dwFilesize);
00716                 if (lpSrvStruct->doneNum < lpSrvStruct->totalNum &&
00717                     FTNextFile(lpSrvStruct, 0)) {
00718                     lpSrvStruct->state = StateFileToSend;
00719                     sendhdr = TRUE;
00720                 } else if (shdr->bFlags & FLAGS_CONT) {
00721                     ShowStatusWindow(rTicket, SW_HIDE);
00722                     lpSrvStruct->state = StateListWantToGet;
00723                     lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
00724                     SET_RENDEZVOUS_DONE(rTicket);
00725                     SockStartWaitTimer(rTicket);
00726                     return 0;
00727                 } else {
00728                     SET_RENDEZVOUS_DONE(rTicket);
00729                     return 1;
00730                 }
00731             }
00732         } else if (hdrType == HDR_TYPE_LIST_TO_SEND) {
00733             if (state == StateListToSend) {
00734                 lpSrvStruct->totalNum = SWAP2(shdr->wTotalNumFiles);
00735                 lpSrvStruct->totalSize = SWAP4(shdr->dwTotalFilesize);
00736                 if (rTicket->cmdID == CMDID_GET_LIST) {
00737                     FTSetDirPath(rTicket,0,TRUE);
00738                     if (lpSrvStruct->fileP)
00739                         fclose(lpSrvStruct->fileP);
00740                     lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"wb");
00741                     if (!lpSrvStruct->fileP) {
00742                         LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
00743                             buf,sizeof buf);
00744                         MessageBox(rTicket->hDlgWnd, buf, 0, MB_OK);
00745                         return 1;
00746                     }
00747                     lpSrvStruct->state = StateListOkToSend;
00748                     sendhdr = TRUE;
00749                 }
00750             }
```

```
00751        } else if (hdrType == HDR_TYPE_LIST_OK_TO_SEND) {
00752            if (state == StateListOkToSend) {
00753                lpSrvStruct->state = StateListData;
00754                sendhdr = FALSE;
00755            }
00756        } else if (hdrType == HDR_TYPE_LIST_FOOTER) {
00757            if (state == StateListFooter) {
00758                AppendMsg(rTicket,IDSFTP_RecvrDone,FALSE,
00759                          ISFT_RCVR(lpSrvStruct),TRUE);
00760                SET_RENDEZVOUS_DONE(rTicket);
00761                if (shdr->bFlags & FLAGS_CONT) {
00762                    ShowStatusWindow(rTicket, SW_HIDE);
00763                    lpSrvStruct->state = StateListWantToGet;
00764                    lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
00765                    SockStartWaitTimer(rTicket);
00766                    return 0;
00767                }
00768            }
00769        } else if (hdrType == HDR_TYPE_LIST_WANT_TO_GET) {
00770            if (state == StateListWantToGet) {
00771                FTReInitHdr(rTicket);
00772                lpSrvStruct->state = StateFileToSend;
00773                ShowStatusWindow(rTicket, SW_SHOW);
00774                FTPrepareForPut(rTicket, shdr->bName);
00775                sendhdr = TRUE;
00776                lpSrvStruct->sock_timeout = 0;
00777            }
00778        }
00779        if (sendhdr == -1)
00780            return 1;
00781
00782        SockSend(rTicket,sendhdr);
00783        return 0;
00784 }
00785
00786 void FTIncrementState(LPRENDEZVOUSTICKET rTicket)
00787 {
00788        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00789        FTHDR *shdr = &lpSrvStruct->sock_hdr;
00790        int state = lpSrvStruct->state;
00791
00792 // There are 3 state paths that are followed;
00793 // 1) FToSend->FOkToSend->FData->FFooter
00794 // 2) FToSend->FWantToResume->FToResume->FOkToResume->FData->FFooter
00795 // 3) LToSend->LOkToSend->LData->LFooter/LWantToGet->FToSend...
00796        switch (state) {
00797            case StateFileToSend:
00798                lpSrvStruct->state = StateFileOkToSend;
00799                break;
00800            case StateFileOkToSend:
00801                lpSrvStruct->state = StateFileData;
00802                break;
00803            case StateFileData:
00804                lpSrvStruct->state = StateFileFooter;
00805                if (rTicket->cmdID == CMDID_GET_LIST)
00806                    shdr->bFlags |= FLAGS_CONT;
00807                break;
00808            case StateFileFooter:
00809                if (rTicket->cmdID == CMDID_GET_LIST)
00810                    lpSrvStruct->state = StateListWantToGet;
00811                else
00812                    lpSrvStruct->state = StateFileToSend;
00813                break;
```

```
00814          case StateFileWantToResume:
00815            lpSrvStruct->state = StateFileToResume;
00816            break;
00817          case StateFileToResume:
00818            lpSrvStruct->state = StateFileOkToResume;
00819            break;
00820          case StateFileOkToResume:
00821            lpSrvStruct->state = StateFileData;
00822            break;
00823          case StateListToSend:
00824            lpSrvStruct->state = StateListOkToSend;
00825            break;
00826          case StateListOkToSend:
00827            lpSrvStruct->state = StateListData;
00828            break;
00829          case StateListData:
00830            lpSrvStruct->state = StateListFooter;
00831            shdr->bFlags |= FLAGS_CONT;
00832            break;
00833          case StateListFooter:
00834            lpSrvStruct->state = StateListWantToGet;
00835            break;
00836          case StateListWantToGet:
00837            lpSrvStruct->state = StateFileToSend;
00838            break;
00839        }
00840 }
00841
00842 BOOL FTCountFilesToSend(LPSTR path, DWORD* totsize, WORD* totnum)
00843 {
00844      WIN32_FIND_DATA ffData;
00845      HANDLE ffh = FindFirstFile(path,&ffData);
00846      BOOL isdir = FALSE;
00847
00848      // TODO: how to tell path is a dir and not a * cmd?  isdir = TRUE;
00849      *totnum = 0;
00850      *totsize = 0;
00851      if (ffh != INVALID_HANDLE_VALUE) {
00852        do {
00853          if (ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
00854              // char dirpath[MAX_PATH];
00855              // TODO: go into subdirs also  (recursively call this func)
00856          } else {
00857              *totnum += 1;
00858              *totsize += ffData.nFileSizeLow;
00859          }
00860        } while (FindNextFile(ffh,&ffData));
00861        FindClose(ffh);
00862      }
00863      return isdir;
00864 }
00865
00866 void FTMakeLocalPath(LPRENDEZVOUSTICKET rTicket, LPSTR localpath, int size)
00867 {
00868      LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);
00869      GetDirDownload(localpath);
00870      MakeDir(localpath);
00871      if (rTicket->cmdID == CMDID_SEND_FILE) {
00872          LPSTR path = &lpSrvProFt->bName[0];
00873          LPSTR lp, lpe, cp = _fstrrchr(path,'\\');
00874          if (!cp)
00875              cp = _fstrrchr(path,'/');    // in case its Unix
00876          if (!cp)
```

```
00877                cp = _fstrrchr(path,':');    // in case its Mac
00878            if (!cp)
00879                cp = path;
00880            else
00881                cp++;
00882            for (lp=localpath; *lp; lp++);
00883            if (*(lp-1) != '\\')
00884                *lp++ = '\\';
00885            for (lpe=localpath+size-1; *cp && lp < lpe; )
00886                *lp++ = *cp++;
00887            *lp = '\0';
00888        }
00889 }
00890
00891 BOOL FTPrepareForPut(LPRENDEZVOUSTICKET rTicket, LPSTR fp)
00892 {
00893        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00894        BOOL isDir;
00895        WORD numFiles;
00896        DWORD totSize;
00897        char filelib[2*MAX_PATH];
00898
00899        GetDirFilelib(filelib);
00900
00901        if (*fp == '\0') {
00902            lpSrvStruct->state = StateListToSend;
00903            lpSrvStruct->sorted = FTConstructDirListing(filelib, &totSize,
00904                                                        &numFiles);
00905        } else if (CheckForSecurityHoles(fp)) {
00906            AppendMsg(rTicket, IDSFTP_RecvrDecline, FALSE, FALSE, FALSE);
00907            SET_RENDEZVOUS_DECLINE(rTicket);
00908            CleanUp(rTicket, 0);
00909            return FALSE;
00910        } else {
00911            LPSTR cp = filelib + lstrlen(filelib);
00912            lstrcpy(cp,fp);
00913            lpSrvStruct->state = StateFileToSend;
00914            isDir = FTCountFilesToSend(filelib, &totSize, &numFiles);
00915        }
00916        // if there are no files in the FILE_LIBRARY, return an IGNORE NAK
00917        if (!numFiles) {
00918            AppendMsg(rTicket, IDSFTP_HasNoFiles, FALSE, TRUE, FALSE);
00919            SET_RENDEZVOUS_IGNORE(rTicket);
00920            CleanUp(rTicket, 0);
00921            return FALSE;
00922        }
00923        lpSrvStruct->totalNum = numFiles;
00924        lpSrvStruct->totalSize = totSize;
00925        FTNextFile(lpSrvStruct, filelib);
00926        FTSetDirPath(rTicket,filelib,0);
00927        return TRUE;
00928 }
00929
00930 void FTEnableGetBut(HWND hwndList, HWND hwndGet)
00931 {
00932        //int selid = (int)SendMessage(hwndList, LB_GETCURSEL,0,0);
00933        //EnableWindow(hwndGet, selid != LB_ERR);
00934        int n = (int)SendMessage(hwndList, LB_GETSELCOUNT,0,0);
00935        EnableWindow(hwndGet, (n > 0));
00936 }
00937
00938 void FTGetListItem(LPRENDEZVOUSTICKET rTicket)
00939 {
```

```
00940        char oneline[MAX_LIST_LINE];
00941        LPSTR cp;
00942        DWORD num;
00943        int selid,n;
00944        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
00945        FTHDR *shdr = &lpSrvStruct->sock_hdr;
00946        HWND hwndDlg  = rTicket->hDlgWnd;
00947        HWND hwndList = GetDlgItem(hwndDlg, IDC_DIR_FILELIST);
00948        HWND hwndGet  = GetDlgItem(hwndDlg, IDC_GET);
00949        HWND hwndStop = GetDlgItem(hwndDlg, IDABORT);
00950
00951        //selid = (int)SendMessage(hwndList, LB_GETCURSEL,0,0);
00952        //if (selid != LB_ERR) {
00953        n = (int)SendMessage(hwndList, LB_GETSELCOUNT,0,0);
00954        if (n > 0) {
00955            SendMessage(hwndList,LB_GETSELITEMS,1,(LPARAM)(LPSTR)&selid);
00956            lpSrvStruct->selID = selid;
00957            SendMessage(hwndList,LB_GETTEXT, selid, (LPARAM)(LPSTR)oneline);
00958            cp = oneline + shdr->bListNameOffset + 1; // one for extra " " at
begin
00959            lstrcpy(&shdr->bName[0],cp);
00960            FTReInitHdr(rTicket);
00961            cp = oneline + shdr->bListSizeOffset + 1; // one for extra " " at
begin
00962            num = (DWORD)atol(cp);
00963            shdr->dwFilesize = SWAP4(num);
00964            shdr->dwTotalFilesize = shdr->dwFilesize;
00965            lpSrvStruct->sock_timeout = 0;
00966            SockSend(rTicket,TRUE);
00967
00968            ShowThermo(rTicket->hDlgWnd,SW_SHOW,SW_HIDE);
00969            SetFocus(hwndList);
00970            EnableWindow(hwndGet, FALSE);
00971            EnableWindow(hwndStop, TRUE);
00972            EnableWindow(hwndList, FALSE);
00973        } else {
00974            lpSrvStruct->totalSizeOfDoneFiles = 0;
00975            FTEnableGetBut(hwndList,hwndGet);
00976            EnableWindow(hwndStop, FALSE);
00977            EnableWindow(hwndList, TRUE);
00978            SockStartWaitTimer(rTicket);
00979        }
00980 }
00981
00982 // returns if file is successfully sorted
00983 BOOL FTSortFile(LPSTR path, WORD offset)
00984 {
00985        BOOL ret = FALSE;
00986        // has to be "rb" in order for filelength to work right
00987        FILE *fileP = fopen(path,"rb");;
00988        if (fileP) {
00989            LPSTR fileMem,lp,lpe;
00990            LPSTR* names;
00991            LPSTR* np;
00992            LPSTR* np1;
00993            LPSTR* np2;
00994            int n, nn, nl = 0;
00995            long ntot, nr, size;
00996
00997            size = filelength(fileno(fileP));
00998            if (!size)
00999                return TRUE;
01000            fileMem = (LPSTR)MemAlloc(size + size); // make room for ptrs
```

```
01001          lp = fileMem;
01002          lpe = fileMem + size;
01003          names = (LPSTR*)(lpe+2);
01004          np = names;
01005          ntot = size;
01006          *lpe = '\n';
01007          *(lpe+1) = 0;
01008          if (!fileMem) {
01009              fclose(fileP);
01010              return FALSE;
01011          }
01012          while (ntot) {
01013              nr = ntot;
01014              if (nr > 0x7fff)
01015                  nr = 0x7fff;
01016              if (fread(lp,1,nr,fileP) == 0)
01017                  break;
01018              lp += nr;
01019              ntot -= nr;
01020          }
01021          fclose(fileP);
01022          // fill in pointers to beginning of each row
01023          lp = fileMem;
01024          while (lp < lpe) {
01025              *np++ = lp;
01026              nl++;
01027              while (*lp != '\r' && *lp != '\n')
01028                  lp++;
01029              while (*lp == '\r' || *lp == '\n')
01030                  *lp++ = 0;
01031          }
01032          // now sort pointers; dont worry too much about speed
01033          for (n=nl; n; n--) {
01034              np1 = names;
01035              for (nn=1; nn<n; nn++) {
01036                  LPSTR cp1,cp2;
01037                  np2 = np1 + 1;
01038                  cp1 = *np1 + offset;
01039                  cp2 = *np2 + offset;
01040                  if (lstrcmp(cp1,cp2) > 0) {
01041                      cp1 = *np1;
01042                      *np1 = *np2;
01043                      *np2 = cp1;
01044                  }
01045                  np1++;
01046              }
01047          }
01048          // now write the sorted lines back to disk
01049          fileP = fopen(path,"wb");
01050          if (fileP) {
01051              np1 = names;
01052              while (nl--) {
01053                  int len = lstrlen(*np1);
01054                  fwrite(*np1,len,1,fileP);
01055                  fwrite("\r\n",2,1,fileP);
01056                  np1++;
01057              }
01058              fclose(fileP);
01059              ret = TRUE;
01060          }
01061          MemFree(fileMem);
01062      }
01063      return ret;
```

```
01064 }
01065
01066 void FTInitUnsorted(LPSTR path, HWND hwnd, HWND hwndList, int charwid)
01067 {
01068     FILE *fileP;
01069     char oneline[MAX_LIST_LINE];
01070     RECT wndRect;
01071     int wold, w, wdel;
01072
01073     SendMessage(hwndList, LB_RESETCONTENT, 0, 0);
01074     fileP = fopen(path,"rb");
01075     if (fileP) {
01076         int nr, npos = 0, selid = 0, maxn = 0, len;
01077         oneline[0] = ' ';      // replaced with * when file transferred
01078         while ((nr = fread(&oneline[1],1,sizeof(oneline)-2,fileP)) > 0) {
01079             int nn = 0;
01080             LPSTR line1 = &oneline[1];
01081
01082             while (nn < nr && *line1 != '\r' && *line1 != '\n') {
01083                 line1++;
01084                 nn++;
01085             }
01086             while (*line1 == '\r' || *line1 == '\n') {
01087                 *line1++ = '\0';
01088                 nn++;
01089             }
01090             npos += nn;
01091             fseek(fileP, npos,SEEK_SET);
01092
01093             SendMessage(hwndList,LB_INSERTSTRING,selid++,(LPARAM)&oneline[0]);
01094             len = lstrlen(oneline);
01095             if (len > maxn)
01096                 maxn = len;
01097         }
01098         fclose(fileP);
01099
01100         // expand the window to fit more chars
01101         GetWindowRect(hwndList, &wndRect);
01102         ScreenRectToClient(hwnd, &wndRect);
01103         wold = wndRect.right - wndRect.left;
01104         maxn += 4;   // to allow for scroll bar
01105         w = maxn * charwid;
01106         if (w > wold) {
01107             wdel = w - wold;
01108             MoveWindow(hwndList, wndRect.left, wndRect.top,
01109                         w, wndRect.bottom - wndRect.top, TRUE);
01110             GetWindowRect(hwnd, &wndRect);
01111             wold = wndRect.right - wndRect.left;
01112             MoveWindow(hwnd, wndRect.left, wndRect.top, wold + wdel,
01113                         wndRect.bottom - wndRect.top, TRUE);
01114         }
01115     }
01116 }
01117
01118 void FTInitFileList(LPRENDEZVOUSTICKET rTicket, BOOL firstTime)
01119 {
01120     FILE *fileP = NULL;
01121     char oneline[MAX_LIST_LINE], buf[256];
01122     HWND hwndDlg = rTicket->hDlgWnd;
01123     HWND hwndList = GetDlgItem(hwndDlg, IDC_DIR_FILELIST);
01124     HWND hwndGet  = GetDlgItem(hwndDlg, IDC_GET);
01125     HWND hwndStop = GetDlgItem(hwndDlg, IDABORT);
01126     HWND hwndText = GetDlgItem(hwndDlg, IDC_STATUS_TEXT1);
```

```
01127    HWND hwndTherm= GetDlgItem(hwndDlg, IDC_THERMO);
01128    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
01129    FTHDR *shdr = &lpSrvStruct->sock_hdr;
01130    int selid = 0;
01131
01132    if (firstTime) {
01133        lpSrvStruct->totalSizeOfDoneFiles = 0;
01134        FTSetDirPath(rTicket,0,TRUE);
01135        if (!(shdr->bFlags & FLAGS_SORTED))
01136            FTSortFile(lpSrvStruct->dirPath, shdr->bListNameOffset);
01137        FTInitUnsorted(lpSrvStruct->dirPath, hwndDlg, hwndList,
01138                       lpSrvStruct->listFontWidth);
01139
01140        LoadString(lpOCMInfo->hModule, IDSFT_FileListInfo, buf, sizeof buf);
01141        SetWindowText(hwndText, buf);
01142        ShowThermo(hwndDlg,SW_HIDE,SW_SHOW);
01143        EnableWindow(hwndGet,FALSE);
01144        EnableWindow(hwndStop,FALSE);
01145        SetFocus(hwndList);
01146    } else {
01147        selid = lpSrvStruct->selID;
01148        InvalidateRect(hwndDlg,0,TRUE);
01149        // we just finished transferring the selection; update the list box
01150        SendMessage(hwndList, LB_SETSEL, 0, selid);
01151        SendMessage(hwndList, LB_GETTEXT, selid, (LPARAM)(LPSTR)oneline);
01152        SendMessage(hwndList, LB_DELETESTRING, selid, 0);
01153        oneline[0] = '*';
01154        SendMessage(hwndList, LB_INSERTSTRING, selid, (LPARAM)oneline);
01155        FTGetListItem(rTicket);
01156        return;
01157    }
01158 }
```

```
01492            case IDC_GET:
01493                FTGetListItem(rTicket);
01494                return 1;
```

```
01705 void RequestAndListen(LPRENDEZVOUSTICKET rTicket, int ids, TCHAR* path)
01706 {
01707     TCHAR preBuf[MAX_PRETEXT_SIZE], buf[512];
01708
01709     LoadString(lpOCMInfo->hModule, ids, buf, sizeof(buf));
01710     wsprintf(preBuf, buf, path);
01711     rTicket->preText = preBuf;
01712     rTicket->timeoutTime = GetTickCount() + RENDEZVOUS_TIMEOUT_DEFAULT;
01713     SET_RENDEZVOUS_IPADDR(rTicket);
01714
01715     if (OMSendMessageByRef(OMTYPE_REQUEST, OMGROUP_RENDEZVOUS,
01716                            OMSG_RENDEZVOUS_REQ_PROPOSAL,
01717                            sizeof(RENDEZVOUSTICKET), rTicket)) {
01718         HWND hwndDlg = rTicket->hDlgWnd;
01719         rTicket->hDlgWnd = 0;
01720         DestroyWindow(hwndDlg);
01721         OpenStatus(rTicket,TRUE);
01722         SockListen(rTicket);
01723     }
01724 }
01725
01726 void DoSockConnect(LPRENDEZVOUSTICKET rTicket)
01727 {
01728     OpenStatus(rTicket,FALSE);
01729     SockCleanup(rTicket);
01730     SockConnect(rTicket);
01731 }
01732
01733 void DoStartGet(LPRENDEZVOUSTICKET rTicket, LPSTR path)
01734 {
01735     HWND hwndDlg = rTicket->hDlgWnd;
01736     FILE *InStream = NULL;
01737     HWND hwndFile = GetDlgItem(hwndDlg, IDC_FILE);
01738     int len, ids;
01739     LPSRVPROFT lpSrvProFt;
01740     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
01741
01742     len = lstrlen(path) + sizeof(SRVPROFT);
01743     lpSrvProFt = (LPSRVPROFT)rTicket->lpSrvProposal;
01744     if (!lpSrvProFt) {
01745         CleanUp(rTicket, 0);
01746         return;
01747     }
01748     rTicket->lenSrvProposal = len;
01749     len -= 4;
01750     lpSrvStruct->type = TypeGet;
```

Page 28

```
01751     if (path[0] == '\0') {
01752         lpSrvStruct->state = StateListToSend;
01753         lpSrvProFt->wSubtype = SUBTYPE_GET_LIST;
01754         rTicket->cmdID = CMDID_GET_LIST;
01755         ids = IDSFTP_GetListRequest;
01756     } else {
01757         lpSrvStruct->state = StateFileToSend;
01758         lpSrvProFt->wSubtype = SUBTYPE_GET_FILES;
01759         ids = IDSFTP_GetterRequest;
01760     }
01761     lstrcpy(&lpSrvProFt->bName[0], path);
01762     lpSrvProFt->wTag = SWAP2(RENDEZVOUS_TLV_TAGS_SERVICE_DATA);
01763     lpSrvProFt->wLen = SWAP2(len);
01764     FTMakeLocalPath(rTicket, lpSrvStruct->dirPath,
01765                     sizeof(lpSrvStruct->dirPath));
01766     RequestAndListen(rTicket, ids, path);
01767 }
```

```
02031      if (rTicket->cmdID == CMDID_GET_LIST) {
02032          DoStartGet(rTicket, "");
```

```
02102 BOOL DoStartPut(LPRENDEZVOUSTICKET rTicket)
02103 {
02104     TCHAR buf[512];
02105     LPSTR fp;
02106     int ids;
02107     LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);
02108     if (!lpSrvProFt ||
02109         (lpSrvProFt->wTag != SWAP2(RENDEZVOUS_TLV_TAGS_SERVICE_DATA))) {
02110         SET_RENDEZVOUS_BUSTED(rTicket);
02111         return FALSE;
02112     }
02113
02114     SET_RENDEZVOUS_NO_PROMPT(rTicket);
02115     fp = &lpSrvProFt->bName[0];
02116     if (*fp == '\0')
02117         ids = IDSFTP_GetListRequest;
02118     else
02119         ids = IDSFTP_GetterRequest;
02120
02121     LoadString(lpOCMInfo->hModule, ids, buf, sizeof(buf));
02122     wsprintf(rTicket->preText, buf, fp);
02123     rTicket->lenSrvStruct = sizeof(SRVSTRUCT);
02124     return TRUE;
02125 }
02126
```

Page 34

```
02127 void DoStartPutPost(LPRENDEZVOUSTICKET rTicket)
02128 {
02129     // Note: the lpSrvStruct cannot be used before this
02130     long allow;
02131     LPSRVSTRUCT lpSrvStruct;
02132     LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);
02133
02134     lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
02135     lpSrvStruct->ffh = INVALID_HANDLE_VALUE;
02136
02137     allow = ProfGetLong(PROF_USER, FT_KEY, FT_GET_ALLOW);
02138     if (allow == AllowNoOne ||
02139         (allow == AllowBuddy && BuddyNotOnBuddyList(rTicket->nickname))) {
02140         SET_RENDEZVOUS_DECLINE(rTicket);
02141         CleanUp(rTicket, 0);
02142     } else {
02143         lpSrvStruct->type = TypePut;
02144         if (FTPrepareForPut(rTicket,&lpSrvProFt->bName[0]))
02145             DoSockConnect(rTicket);
02146     }
02147 }
```

```
02430            else if (getCmdActivated && CMDID_IS_GET(rTicket->cmdID))
02431                 return (DoStartPut(rTicket));
```

```
02444          if (CMDID_IS_GET(rTicket->cmdID))
02445              DoStartPutPost(rTicket):
```

```
02500            {
02501                LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
02502                if (rTicket->hModule == lpOCMInfo->hModule) {
02503                    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
02504                    if (!lpSrvStruct || !(lpSrvStruct->sock_flags & SockConnected) ||
02505                        rTicket->reason == OM_PROTO_USER_EXIT) {
02506                        CleanUp(rTicket,0);
02507                    } else {
02508                        char buf[512];
02509                        if (!goingOffline) {
02510                            HWND hwndFocus = GetFocus();
02511                            HWND hwnd = GetTopmostAppWindow();
02512                            LoadString(lpOCMInfo->hModule,IDSFT_offline,
02513                                       buf,sizeof buf);
02514                            DisableAppWindows();
02515                            goingOffline = MessageBox(hwnd, buf, 0,
02516 MB_OKCANCEL|MB_DEFBUTTON1|MB_ICONEXCLAMATION);
02517                            EnableAppWindows();
02518                            SetFocus(hwndFocus);
02519
02520 // when Sesame is finally used...
02521 //                          goingOffline=OkCancelBox(lpOCMInfo->hModule,
02522 //                                                  IDSFT_offline);
02523
02524                        }
02525                        if (goingOffline==IDCANCEL)
02526                            CleanUp(rTicket,0);
02527                    }
02528                    lpInfo->fContinue = FALSE;
02529                }
02530                return 0;  // so goingOffline can be cleared with any other message
02531            }
02532            case OMSG_RENDEZVOUS_EVT_ONLINE:
02533            {
02534                LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
02535                if (rTicket->hModule == lpOCMInfo->hModule) {
02536                    lpInfo->fContinue = FALSE;
02537                }
02538                break;
02539            }
02540            case OMSG_RENDEZVOUS_EVT_ACCEPTED:
02541            {
02542                LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
02543                if (rTicket->hModule == lpOCMInfo->hModule) {
02544                    lpInfo->fContinue = FALSE;
02545                }
02546                break;
02547            }
02548        }
02549        goingOffline = 0;
02550        return 0;
02551 }
02552
02553 //------------------------------------------------------------------------
02554 // Boilerplate OCM stuff
02555 //------------------------------------------------------------------------
02556
02557 BOOL API __export OCMOpen(LPOCMINFO lpOCM)
02558 {
02559     lpOCMInfo = lpOCM;
02560     if(!OMRegister(OMTYPE_EVENT, OMGROUP_RENDEZVOUS, EventHandler))
```
Page 41

```
02940    GROUPBOX        "when others issue the File Get command:" IDC_GROUP2,
02941                    6, 85, 185, 102
02942    CONTROL         "Allow no users to get my files",
02943                    IDC_ALLOW_NOONE,"Button",
02944                    BS_AUTORADIOBUTTON | WS_TABSTOP | WS_GROUP,14,93,170,14
02945    CONTROL         "Allow only users on my Buddy List to get my files",
02946                    IDC_ALLOW_BUDDY,"Button",
02947                    BS_AUTORADIOBUTTON | WS_TABSTOP,14,104,170,14
02948    CONTROL         "Allow everyone to get my files",IDC_ALLOW_ALL,"Button",
02949                    BS_AUTORADIOBUTTON | WS_TABSTOP,14,115,170,14
02950    LTEXT           "Directory from where others can get my files:",
02951                          IDC_STATIC,14,130,160,12
02952    CONTROL         "",IDC_DIR_FILELIB,"Edit",
02953                    ES_AUTOHSCROLL | WS_BORDER | WS_GROUP | WS_TABSTOP,
02954                    14, 141, 171, 13
02955 //                    14, 141, 115, 13
02956 //  PUSHBUTTON       "B&rowse...",IDC_BROWSE_UPLOAD,
02957 //                              135,140,50,14,WS_GROUP | WS_TABSTOP
02958    CONTROL         "&Never display Status dialog",IDC_PUT_NO_STATUS,
02959                          "Button", BS_AUTOCHECKBOX |
WS_TABSTOP,14,155,170,14
02960    CONTROL         "&Keep a record in logfile.txt of who has gotten files",
02961                          IDC_PUT_LOG_FILES,
02962                          "Button", BS_AUTOCHECKBOX |
WS_TABSTOP,14,169,170,14
02963
```

```
03108 FILELISTDLG DIALOG DISCARDABLE  10, 40, 270, 135
03109 STYLE WS_POPUP | WS_DLGFRAME | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
03110 CAPTION "File Listing"
03111 FONT 8, "MS Sans Serif"
03112 BEGIN
03113       CONTROL           "",IDC_THERMO,"Static",SS_SIMPLE | WS_GROUP,8,3,192,39
03114       CTEXT             "",IDC_STATUS_TEXT1,8,12,234,27
03115       CONTROL           "Fast", IDC_IMFT_SPEED_FAST,"Button",
03116                         BS_AUTORADIOBUTTON | WS_TABSTOP | WS_GROUP,208,1,40,12
03117       CONTROL           "Medium", IDC_IMFT_SPEED_MEDIUM,"Button",
03118                         BS_AUTORADIOBUTTON | WS_TABSTOP,208,11,40,12
03119       CONTROL           "Slow", IDC_IMFT_SPEED_SLOW,"Button",
03120                         BS_AUTORADIOBUTTON | WS_TABSTOP,208,21,40,12
03121       CONTROL           "Pause", IDC_IMFT_SPEED_PAUSE,"Button",
03122                         BS_AUTORADIOBUTTON | WS_TABSTOP,208,31,40,11
03123    CONTROL             "",IDC_DIR_FILELIST, LISTBOX,
03124                                 LBS_EXTENDEDSEL|LBS_MULTIPLESEL|
03125                                 LBS_STANDARD|WS_VSCROLL|SBS_HORZ|WS_TABSTOP,
03126                                 8,45,254,68
03127       LTEXT             IDI_SENDFILE_GET, IDC_STATUS_ICON2, 10,114,32,32, SS_ICON
03128       DEFPUSHBUTTON     "Get",IDC_GET,42,117,51,14,WS_GROUP
03129       PUSHBUTTON        "Stop",IDABORT,107,117,51,14,WS_GROUP
03130       PUSHBUTTON        "Cancel",IDCANCEL,172,117,51,14,WS_GROUP
03131 END
```

```
03215    FILETYPE            VFT_DLL
03216    FILESUBTYPE         VFT2_UNKNOWN
03217    FILEVERSION         0, 0, 0, 0
03218
03219 BEGIN
03220     BLOCK "StringFileInfo"
03221     BEGIN
03222         BLOCK VERSION_BUILD_TRANSLATION_STRING
03223         BEGIN
03224             VALUE "CompanyName",           VERSION_COMPANY
03225             VALUE "LegalCopyright",         VERSION_COPYRIGHT
03226             VALUE "ProductName",            VERSION_PRODUCT_NAME
03227             VALUE "ProductVersion",         VERSION_PRODUCT_VERSION_STRING
03228             VALUE "Build Number",           VERSION_BUILD_NUMBER
03229
03230             // Module-specific info
03231             VALUE "FileDescription",        "Icbm File Transfer Module"
03232             VALUE "FileVersion",            "0.0.0.0"
03233             VALUE "InternalName",           "ICBMFT"
03234             VALUE "OriginalFilename",        "ICBMFT.OCM"
03235         END
03236     END
03237
03238     BLOCK "VarFileInfo"
03239     BEGIN
03240         VALUE "Translation",  VERSION_BUILD_TRANSLATION
03241     END
03242 END
03243
03244
03245
03246
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
03247
03248 //
03249 // (C) Copyright 1997  America Online, Inc. 75 Second Ave.
03250 //                      Needham, MA 02194
03251 //
03252
03253 #include "icbmft.h"
03254 #include "string.h"
03255
03256 /* The sequence of events is:
03257     1.  Requester starts a Listen and sends REQUEST to buddy, with timeout
03258     1a. Requester times out --> Cancels
03259     2.  Receiver clicks "Accept" button, starts a Connect
03260     2a. Receiver Connect timesout; starts a Listen and sends ACCEPT to buddy
03261     2b. Sender gets ACCEPT, stops Listen and starts a Connect
03262     2c. Sender or Receiver timesout --> Cancels
03263     3.  Connection completed; start socket protocol
03264 */
03265
03266 void SockQuit(LPRENDEZVOUSTICKET rTicket)
03267 {
03268     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03269     if (IS_RENDEZVOUS_DONE(rTicket)) {
03270     } else
03271         AppendMsg(rTicket,IDSFTP_RecvrCannotConnect,TRUE,FALSE,FALSE);
03272     SockCleanup(rTicket);
03273     CleanUp(rTicket, 0);
03274 }
03275
03276 void SockStartQuitTimer(LPRENDEZVOUSTICKET rTicket)
```

```
03277 {
03278        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03279        if (!lpSrvStruct || IS_RENDEZVOUS_DONE(rTicket)) {
03280            SockQuit(rTicket);
03281            return;
03282        }
03283
03284        // wait five seconds to give us enough time to get a CANCEL or NAK snac
03285        lpSrvStruct->sock_flags |= SockQuiting;
03286        lpSrvStruct->sock_timeout = GetTickCount() + 5000;
03287        o_SetTimer(rTicket->hDlgWnd, 102, 5000, NULL);
03288 }
03289
03290 void SockStartWaitTimer(LPRENDEZVOUSTICKET rTicket)
03291 {
03292        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03293        if (lpSrvStruct) {
03294            // keep connection open 10 minutes
03295            lpSrvStruct->sock_timeout = GetTickCount() + (DWORD)(60000*10);
03296            o_SetTimer(rTicket->hDlgWnd, 102, 60000, NULL);
03297        }
03298 }
03299
03300 void SockListen(LPRENDEZVOUSTICKET rTicket)
03301 {
03302        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03303        if (!lpSrvStruct)
03304            return;
03305
03306        SockCleanup(rTicket);
03307        lpSrvStruct->sock_bufsize = SOCK_BUFSZ;
03308
03309        lpSrvStruct->socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
03310        lpSrvStruct->sin_in.sin_family     = AF_INET;
03311        lpSrvStruct->sin_in.sin_port       = htons(rTicket->port);
03312        lpSrvStruct->sin_in.sin_addr.s_addr = INADDR_ANY;
03313        bind(lpSrvStruct->socket, (struct sockaddr *)&lpSrvStruct->sin_in,
03314            sizeof(lpSrvStruct->sin_in));
03315        WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
03316                    FD_ACCEPT|FD_READ|FD_WRITE|FD_CLOSE);
03317
03318        listen(lpSrvStruct->socket,1);
03319        lpSrvStruct->sock_flags = SockListening;
03320        lpSrvStruct->sock_timeout = 0;  // caller should set this if timeout
needed
03321
03322        // we don't need a timer for requester because it's handled by IM window
03323 }
03324
03325 void SockCleanup(LPRENDEZVOUSTICKET rTicket)
03326 {
03327        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03328        if (lpSrvStruct) {
03329            if (lpSrvStruct->sock_flags &
SockListening|SockConnecting|SockConnected) {
03330                WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, 0, 0);
03331                closesocket(lpSrvStruct->socket);
03332                lpSrvStruct->socket = 0;
03333            }
03334            lpSrvStruct->sock_flags = 0;
03335        }
03336 }
03337
```

```
03338 void SockAcceptReady(LPRENDEZVOUSTICKET rTicket)
03339 {
03340     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03341     if (!lpSrvStruct)
03342         return;   -
03343
03344     if (lpSrvStruct->sock_flags & SockListening) {
03345         fd_set socks;
03346         struct timeval imtimeout;
03347         FD_ZERO(&socks);
03348         FD_SET(lpSrvStruct->socket,&socks);
03349         imtimeout.tv_sec=0; imtimeout.tv_usec=0;
03350         if (select(0,&socks,0,0,&imtimeout)) {
03351             int mm = sizeof(lpSrvStruct->sin_in);
03352             SOCKET soc;
03353             soc = accept(lpSrvStruct->socket,
03354                         (struct sockaddr *)&lpSrvStruct->sin_in, &mm);
03355             if (soc == INVALID_SOCKET) {
03356                 SockQuit(rTicket);
03357                 return;
03358             }
03359             closesocket(lpSrvStruct->socket);
03360             lpSrvStruct->socket = soc;
03361             lpSrvStruct->sock_flags &= ~SockListening;
03362             WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
03363                         FD_READ|FD_WRITE|FD_CLOSE);
03364             lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
03365             rTicket->timeoutTime = 0;
03366
03367             if (lpSrvStruct->type == TypeSend || lpSrvStruct->type == TypePut)
03368                 SockSend(rTicket,TRUE);
03369             else
03370                 SockRecvReady(rTicket);
03371         }
03372     }
03373 }
03374
03375 void SockRecvReady(LPRENDEZVOUSTICKET rTicket)
03376 {
03377     int n, num;
03378     DWORD nRecvd,chksum;
03379     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03380     FTHDR *shdr = &lpSrvStruct->sock_hdr;
03381     if (!lpSrvStruct)
03382         return;
03383
03384     lpSrvStruct->sock_flags |= SockReadyToReceive;
03385     if (lpSrvStruct->sock_flags & SockIsReceiving ||
03386         !(lpSrvStruct->sock_flags & SockConnected))
03387         return;
03388     lpSrvStruct->sock_flags |= SockIsReceiving;
03389     while (lpSrvStruct->sock_flags & SockReadyToReceive) {
03390         lpSrvStruct->sock_flags &= ~SockReadyToReceive;
03391         if (lpSrvStruct->sock_flags & SockReadyToReceiveHdr) {
03392             char buf[1024];
03393             LPSTR inptr;
03394             lpSrvStruct->speed_iter = 0;
03395             lpSrvStruct->speed_timestart = 0;
03396             if (!(lpSrvStruct->sock_flags & SockRecvingHdr)) {
03397                 inptr = (LPSTR)shdr;
03398                 num = sizeof(FTHDR);
03399             } else {
03400                 // should never get here, unless other client has a filename
```

```
03401                    // >256; throw these extra bytes away since filenames are
03402                    // limited to 256 chars anyway
03403                    inptr = buf;
03404                    num = SWAP2(shdr->wHdrLen) - sizeof(FTHDR);
03405                    if (num > sizeof(buf))
03406                        goto QUIT;
03407                }
03408
03409 //zzz TODO if FLAGS_ABORT, ignore non-header records
03410                n = recv(lpSrvStruct->socket, inptr, num, 0);
03411                if (n == SOCKET_ERROR) {
03412                    int wsaerr = WSAGetLastError();
03413                    if (wsaerr == WSAEWOULDBLOCK)
03414                        break;
03415                    goto QUIT;
03416                } else if ((!(lpSrvStruct->sock_flags & SockRecvingHdr) &&
03417                            n < MIN_HDR_SZ) ||
03418                           ((lpSrvStruct->sock_flags & SockRecvingHdr) &&
03419                            n < num)) {
03420                    goto QUIT;
03421                } else if (!(lpSrvStruct->sock_flags & SockRecvingHdr)) {
03422                    if (FTValidateHdr(rTicket))
03423                        goto QUIT;
03424                    if (n < SWAP2(shdr->wHdrLen)) {
03425                        lpSrvStruct->sock_flags |= 2430-2431
2444-2445SockRecvingHdr;
03426                        continue;
03427                    }
03428                }
03429                lpSrvStruct->sock_flags &=
~(SockRecvingHdr|SockReadyToReceiveHdr);
03430                shdr->bName[FNSZ-1]=0;
03431                if (FTProcessHdr(rTicket))
03432                    goto QUIT;
03433
03434            } else if (lpSrvStruct->sock_flags & SockConnected) {
03435                // We are reading the file here
03436                long todo = SWAP4(shdr->dwFilesize) - SWAP4(shdr->dwNumRecvd);
03437                if (!todo)
03438                    break;
03439                if (lpSrvStruct->status_numTodo == 0) {
03440                    lpSrvStruct->status_numDone = 0;
03441                    lpSrvStruct->status_numTodo = todo;
03442                    lpSrvStruct->sock_starttime = GetTickCount();
03443                    PaintThermo(rTicket,FALSE);
03444                }
03445                if (lpSrvStruct->speed == SpeedPause)
03446                    break;
03447                else if (lpSrvStruct->speed_timewait) {
03448                    DWORD delta = GetTickCount()-lpSrvStruct->speed_timestart;
03449                    if (delta < lpSrvStruct->speed_timewait) {
03450                        delta = (lpSrvStruct->speed_timewait - delta);
03451                        if (delta > 0x7fff)
03452                            delta = 0x7fff;
03453                        lpSrvStruct->sock_flags |= SockRecvDelay;
03454                        o_SetTimer(rTicket->hDlgWnd, 102, delta, NULL);
03455                        break;
03456                    }
03457                }
03458
03459                num = (todo > SOCK_BUFSZ) ? SOCK_BUFSZ : (int)todo;
03460                n = recv(lpSrvStruct->socket, lpSrvStruct->sock_buf, num, 0);
03461                if (n == SOCKET_ERROR) {
```

```
03462                    int wsaerr = WSAGetLastError();
03463                    if (wsaerr == WSAEWOULDBLOCK)
03464                        break;
03465                    SockStartQuitTimer(rTicket);
03466                    return;
03467                } else if (n <= 0) {
03468                    goto QUIT;
03469                }
03470                // write out what we just read
03471                if (fwrite(lpSrvStruct->sock_buf,n,1,lpSrvStruct->fileP) != 1) {
03472                    // TODO: put up error-writing-file message
03473                    goto QUIT;
03474                }
03475
03476                nRecvd = SWAP4(shdr->dwNumRecvd) + n;
03477                shdr->dwNumRecvd = SWAP4(nRecvd);
03478                lpSrvStruct->sock_numSent = nRecvd;
03479                lpSrvStruct->sock_numTotal += n;
03480                lpSrvStruct->status_numDone += n;
03481                chksum = (shdr->dwRecvdChecksum);
03482                chksum = FTRecalcChecksum(chksum,(LPWORD)lpSrvStruct->sock_buf,n);
03483                shdr->dwRecvdChecksum = (chksum);
03484 //zzz TODO if FLAGS_ABORT, pretend it's at the end
03485                if (lpSrvStruct->speed_iter == 0 || lpSrvStruct->speed_timewait) {
03486                    lpSrvStruct->speed_timestart = GetTickCount();
03487                } else if (lpSrvStruct->speed_iter == SPEED_NUM_ITERS  &&
03488                           lpSrvStruct->speed_timewait == 0) {
03489                    DWORD delta = (GetTickCount() -
03490                                        lpSrvStruct->speed_timestart) /
SPEED_NUM_ITERS;
03491                    lpSrvStruct->speed_timefor1 = delta;
03492                    if (lpSrvStruct->speed == SpeedMedium)
03493                        lpSrvStruct->speed_timewait = delta * SPEED_MEDIUM;
03494                    else if (lpSrvStruct->speed == SpeedSlow)
03495                        lpSrvStruct->speed_timewait = delta * SPEED_SLOW;
03496                }
03497                lpSrvStruct->speed_iter++;
03498
03499                PaintThermo(rTicket,FALSE);
03500                if (SWAP4(shdr->dwNumRecvd) == SWAP4(shdr->dwFilesize)) {
03501                    BOOL listData = FALSE;
03502                    // finished with entire file; get ready for next one
03503                    if (shdr->dwChecksum != 0 &&
03504                        shdr->dwChecksum != shdr->dwRecvdChecksum)
03505                        AppendMsg(rTicket,IDSFTP_RecvrDoneBadSum,FALSE,
03506                                    ISFT_RCVR(lpSrvStruct),TRUE);
03507                    else
03508                        AppendMsg(rTicket,IDSFTP_RecvrDone,FALSE,
03509                                    ISFT_RCVR(lpSrvStruct),TRUE);
03510                    FTCloseFileAndSetTime(rTicket);
03511                    listData = (lpSrvStruct->state == StateListData);
03512                    FTIncrementState(rTicket);
03513                    SockSend(rTicket,TRUE);
03514                    lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
03515                    lpSrvStruct->doneNum++;
03516                    lpSrvStruct->totalSizeOfDoneFiles += SWAP4(shdr->dwFilesize);
03517                    if (lpSrvStruct->state == StateListWantToGet) {
03518                        FTInitFileList(rTicket, listData);
03519                        SET_RENDEZVOUS_DONE(rTicket);
03520                    } else if (lpSrvStruct->doneNum >= lpSrvStruct->totalNum &&
03521                        lpSrvStruct->state != StateListWantToGet) {
03522                        SET_RENDEZVOUS_DONE(rTicket);
03523                        goto QUIT;
```

```
03524                    }
03525                }
03526            }
03527        }
03528        lpSrvStruct->sock_flags &= ~SockIsReceiving;
03529        return;
03530 QUIT:
03531        SockQuit(rTicket);
03532        return;
03533 }
03534
03535 void SockConnectComplete(LPRENDEZVOUSTICKET rTicket)
03536 {
03537        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03538        if (!lpSrvStruct)
03539            return;
03540
03541        // MessageBox(0,"SockConnectComplete.",0,MB_OK);   // zzz
03542        lpSrvStruct->sock_timeout = 0;
03543        lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
03544        rTicket->timeoutTime = 0;
03545        if (lpSrvStruct->type == TypeSend || lpSrvStruct->type == TypePut)
03546            SockSend(rTicket,TRUE);
03547        else
03548            SockRecvReady(rTicket);
03549 }
03550
03551 // return FALSE if we cannot connect
03552 BOOL SockConnect(LPRENDEZVOUSTICKET rTicket)
03553 {
03554        LPBYTE ipaddr = rTicket->ipAddrRemoteVerified;
03555        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03556        if (!lpSrvStruct)
03557            goto err;
03558
03559        if (ipaddr[0] == 0 || lpSrvStruct->sock_flags & SockTriedUnverified) {
03560            ipaddr = rTicket->ipAddrRemote;
03561            lpSrvStruct->sock_flags |= SockTriedUnverified;
03562        }
03563        if (ipaddr[0] == 0)
03564            goto err;
03565
03566        lpSrvStruct->sock_bufsize = SOCK_BUFSZ;
03567        lpSrvStruct->sock_timeout = 0;
03568
03569        if (!(lpSrvStruct->sock_flags & SockConnected)) {
03570            struct sockaddr_in addr;
03571            int ret;
03572
03573            rTicket->timeoutTime = GetTickCount() + 60000;   // 1 minute
03574
03575            addr.sin_family = AF_INET;
03576            addr.sin_port   = htons(rTicket->port);
03577            _fmemcpy(&addr.sin_addr, ipaddr, 4);
03578            if (lpSrvStruct->socket) {
03579                WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, 0, 0);
03580                closesocket(lpSrvStruct->socket);
03581            }
03582            lpSrvStruct->socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
03583            WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
03584                        FD_CONNECT|FD_READ|FD_WRITE|FD_CLOSE);
03585            ret = connect(lpSrvStruct->socket, (LPSOCKADDR)&addr, sizeof(addr));
03586            if (ret == SOCKET_ERROR) {
```

```
03587                    int wsaerr = WSAGetLastError();
03588                    if (wsaerr == WSAEWOULDBLOCK) {
03589                        // start timer
03590                        lpSrvStruct->sock_flags |= SockConnecting;
03591                        lpSrvStruct->sock_timeout = GetTickCount() + 20000;
03592                        o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
03593                        return TRUE;
03594                    }
03595                    goto err;
03596                }
03597            lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
03598            rTicket->timeoutTime = 0;
03599            lpSrvStruct->sock_timeout = 0;
03600        }
03601        return TRUE;
03602 err:
03603        SockQuit(rTicket);
03604        return FALSE;
03605 }
03606
03607 void SockTimeout(LPRENDEZVOUSTICKET rTicket)
03608 {
03609        DWORD ticks = GetTickCount();
03610        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03611        if (!lpSrvStruct)
03612            return;
03613
03614        if (lpSrvStruct->sock_timeout && (lpSrvStruct->sock_timeout < ticks))
03615        {
03616            if (lpSrvStruct->sock_flags & (SockClosing|SockQuiting)) {
03617            } else if (IS_RENDEZVOUS_TARGET(rTicket)) {
03618                if (lpSrvStruct->sock_flags & SockConnecting) {
03619                    // cannot connect; start listen and ask buddy to connect to us
03620                    SockListen(rTicket);
03621                    SendCounter(rTicket);
03622                    lpSrvStruct->sock_timeout = GetTickCount() + 20000;
03623                    o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
03624                    return;
03625                }
03626            }
03627            SockQuit(rTicket);
03628        } else if (lpSrvStruct->sock_timeout) {
03629            o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
03630        } else if (lpSrvStruct->sock_flags & SockSendDelay) {
03631            lpSrvStruct->sock_flags &= ~SockSendDelay;
03632            SockSendReady(rTicket);
03633        } else if (lpSrvStruct->sock_flags & SockRecvDelay) {
03634            lpSrvStruct->sock_flags &= ~SockRecvDelay;
03635            SockRecvReady(rTicket);
03636        }
03637 }
03638
03639 void SockSend(LPRENDEZVOUSTICKET rTicket, BOOL sendhdr)
03640 {
03641        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03642        if (!lpSrvStruct)
03643            return;
03644
03645        if (!(lpSrvStruct->sock_flags & SockConnected))
03646            return;  // not connected, so we cannot send anything
03647
03648        lpSrvStruct->sock_flags |= SockSending;
03649        if (sendhdr) {
```

```
03650            lpSrvStruct->sock_flags |= SockSendingHdr;
03651            FTInitHdr(rTicket);
03652        }
03653        SockSendReady(rTicket);
03654 }
03655
03656 void SockSendReady(LPRENDEZVOUSTICKET rTicket)
03657 {
03658        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03659        FTHDR *shdr = &lpSrvStruct->sock_hdr;
03660        if (!lpSrvStruct)
03661            return;
03662
03663        if (!(lpSrvStruct->sock_flags & (SockSendingHdr|SockSending))) {
03664            return;
03665        }
03666        while (1) {
03667            int n, num;
03668            if (lpSrvStruct->sock_flags & SockSendingHdr) {
03669                lpSrvStruct->speed_iter = 0;
03670                lpSrvStruct->speed_timestart = 0;
03671                num = SWAP2(shdr->wHdrLen);
03672                n = send(lpSrvStruct->socket, (LPSTR)shdr,num,0);
03673                if (n == SOCKET_ERROR) {
03674                    int wsaerr = WSAGetLastError();
03675                    if (wsaerr == WSAEWOULDBLOCK)
03676                        return;
03677                    goto QUIT;
03678                } else if (n < num) {
03679                    goto QUIT;
03680                }
03681                break;
03682            } else if (lpSrvStruct->sock_flags & SockSending) {
03683                long lnum = SWAP4(shdr->dwFilesize) -
03684                    lpSrvStruct->sock_numSent;
03685                if (lpSrvStruct->status_numTodo == 0) {
03686                    lpSrvStruct->status_numDone = 0;
03687                    lpSrvStruct->status_numTodo = lnum;
03688                    lpSrvStruct->sock_starttime = GetTickCount();
03689                }
03690 again:
03691                num = (lnum > (long)lpSrvStruct->sock_bufsize) ?
03692                    lpSrvStruct->sock_bufsize : (int)lnum;
03693
03694                if (num) {
03695                    fseek(lpSrvStruct->fileP, lpSrvStruct->sock_numSent,
SEEK_SET);
03696
03697                    if (fread(lpSrvStruct->sock_buf,num,1,lpSrvStruct->fileP) < 1)
03698                        goto QUIT;
03699
03700                    if (lpSrvStruct->speed == SpeedPause)
03701                        return;
03702                    else if (lpSrvStruct->speed_timewait) {
03703                        DWORD delta = GetTickCount()-lpSrvStruct->speed_timestart;
03704                        if (delta < lpSrvStruct->speed_timewait) {
03705                            delta = (lpSrvStruct->speed_timewait - delta);
03706                            if (delta > 0x7fff)
03707                                delta = 0x7fff;
03708                            lpSrvStruct->sock_flags |= SockSendDelay;
03709                            o_SetTimer(rTicket->hDlgwnd, 102, delta, NULL);
03710                            return;
03711                        }
```

```
03712                          }
03713                          n = send(lpSrvStruct->socket, lpSrvStruct->sock_buf, num, 0);
03714                          if (n == SOCKET_ERROR) {
03715                              int wsaerr = WSAGetLastError();
03716                              if (wsaerr == WSAEWOULDBLOCK)
03717                                  return;
03718                              else if (wsaerr == WSAEMSGSIZE &&
03719                                      lpSrvStruct->sock_bufsize > 16) {
03720                                  // 16 above prevents infinite loops
03721                                  lpSrvStruct->sock_bufsize >>= 1;
03722                                  goto again;
03723                              }
03724                              goto QUIT;
03725                          } else {
03726                              lpSrvStruct->sock_numSent += n;
03727                              lpSrvStruct->sock_numTotal += n;
03728                              lpSrvStruct->status_numDone += n;
03729                              if (lpSrvStruct->speed_iter == 0 ||
03730                                      lpSrvStruct->speed_timewait) {
03731                                  lpSrvStruct->speed_timestart = GetTickCount();
03732                              } else if (lpSrvStruct->speed_iter == SPEED_NUM_ITERS &&
03733                                      lpSrvStruct->speed_timewait == 0) {
03734                                  DWORD delta = ((GetTickCount() -
03735                                              lpSrvStruct->speed_timestart) /
03736                                              SPEED_NUM_ITERS);
03737                                  lpSrvStruct->speed_timefor1 = delta;
03738                                  if (lpSrvStruct->speed == SpeedMedium)
03739                                      lpSrvStruct->speed_timewait = delta *
SPEED_MEDIUM;
03740                                  else if (lpSrvStruct->speed == SpeedSlow)
03741                                      lpSrvStruct->speed_timewait = delta * SPEED_SLOW;
03742                              }
03743                              lpSrvStruct->speed_iter++;
03744                          }
03745                          PaintThermo(rTicket,FALSE);
03746                      }
03747                      if (lpSrvStruct->sock_numSent == SWAP4(shdr->dwFilesize))
03748                          break;
03749              } else
03750                  return;
03751          }
03752
03753      lpSrvStruct->sock_flags &= ~(SockSendingHdr|SockSending);
03754      FTIncrementState(rTicket);
03755      if (lpSrvStruct->state != StateFileData &&
03756              lpSrvStruct->state != StateListData)
03757          lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
03758      return;  // next wait for a reply
03759
03760 QUIT:
03761      SockStartQuitTimer(rTicket);
03762      return;
03763
03764 }
03765
03766 void SockConnectionClosed(LPRENDEZVOUSTICKET rTicket)
03767 {
03768      // delay before calling SockQuit() to give time to receive a NAK or CANCEL
03769      LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03770      if (lpSrvStruct) {
03771          lpSrvStruct->sock_flags |= SockClosing;
03772          SockStartQuitTimer(rTicket);
03773      }
```

```
03774 }
03775
03776 LRESULT SockMessage(LPRENDEZVOUSTICKET rTicket, WPARAM wParam, LPARAM lParam)
03777 {
03778     LRESULT result = 1;
03779
03780     WORD event = WSAGETSELECTEVENT(lParam);
03781     WORD error = WSAGETSELECTERROR(lParam);
03782
03783     LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03784     if (lpSrvStruct && (lpSrvStruct->sock_flags & SockClosing)) {
03785         // ignore all socket events because we are waiting for timeout
03786     }
03787     else if (error == 0)
03788     {
03789         if (event & FD_ACCEPT)
03790             SockAcceptReady(rTicket);
03791         if (event & FD_CONNECT)
03792             SockConnectComplete(rTicket);
03793         if (event & FD_READ)
03794             SockRecvReady(rTicket);
03795         if (event & FD_WRITE)
03796             SockSendReady(rTicket);
03797         if (event & FD_CLOSE)
03798             SockConnectionClosed(rTicket);
03799     }
03800     else
03801     {
03802         LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
03803         switch (error)
03804         {
03805           case WSAECONNREFUSED:
03806           case WSAECONNABORTED:
03807           case WSAECONNRESET:
03808           case WSAENETDOWN:
03809           default:
03810             {
03811                 int flags = lpSrvStruct->sock_flags;
03812                 if (!(flags & (SockTriedUnverified|SockListening)) &&
03813                     (flags & SockConnecting)) {
03814                     lpSrvStruct->sock_flags |= SockTriedUnverified;
03815                     SockConnect(rTicket);
03816                 } else if (!(lpSrvStruct->sock_flags & SockConnected)) {
03817                     lpSrvStruct->sock_timeout = GetTickCount() - 1;
03818                     SockTimeout(rTicket);
03819                 } else {
03820                     SockStartQuitTimer(rTicket);
03821                 }
03822                 result = 0;
03823             }
03824         }
03825     }
03826     return result;
03827 }
```

```
00171 void connLookupHost(LPCONNECTION c, LPCSTR host, LPHOSTENT hostent)
00172 {
00173     /* If the host is specified with an IP address, stash the address into
00174      * the hostent buffer and fake a lookup-complete event.
00175      */
00176     if (host[0] >= '0' && host[0] <= '9')
00177     {
00178         DWORD iaddr;
00179
00180         if ((iaddr = inet_addr(host)) == INADDR_NONE)
00181         {
00182             ConnDisconnect(c);
00183             connCallFatalError(c, c->session, IDS_BAD_ADDRESS, NULL);
00184             return;
00185         }
00186
00187         connBuildFakeHostEnt(hostent, host, iaddr);
00188         connEventLookupComplete(c);
```

Page 3

Exhibit 3

```
00189      }
00190
00191      /* If host was specified with a hostname, initiate an asynchronous
00192       * DNS lookup.  This will culminate with either an EventLookupComplete()
00193       * or an EventLookupFailed() call.  Only do a timeout in the case of
00194       * a proxy lookup.  A timeout in the case of a host lookup does not
00195       * work with the "auto dial" feature of Windows 95; it does not allow
00196       * enough time to dial and make the connection.  Also, the error message
00197       * for DNS lookup is confusing to users.
00198       */
00199      else
00200      {
00201          if (c->state == CONN_STATE_PROXY_LOOKUP)
00202              o_SetTimer(c->hSockWnd, TIMER_ID_LOOKUP, TIMEOUT_LOOKUP, NULL);
00203          c->hLookupTask = WSAAsyncGetHostByName(c->hSockwnd,
00204                                                 WM_GETHOSTBYNAME, host,
00205                                                 (LPBYTE)hostent,
00206                                                 MAXGETHOSTSTRUCT);
00207      }
00208 }
```

```
00248 void connConnectToHost(LPCONNECTION c, LPVOID ipaddr, WORD port)
00249 {
00250      SOCKADDR_IN addr;
00251
```

```
00252       connDestroySocket(c);
00253       connCreateSocket(c);
00254
00255       addr.sin_family = AF_INET;
00256       addr.sin_port = htons(port);
00257       _fmemcpy(&addr.sin_addr, ipaddr, 4);
00258       connect(c->sock, (LPSOCKADDR)&addr, sizeof(addr));
00259       o_SetTimer(c->hSockwnd, TIMER_ID_CONNECT, TIMEOUT_CONNECT, NULL);
00260
00261       TRACECONNECT(inet_ntoa(addr.sin_addr), port, c->sock);
00262 }
```

```
00491 void connDoServerLookup(LPCONNECTION c)
00492 {
00493     connChangeState(c, CONN_STATE_LOOKUP);
00494     connLookupHost(c, c->serv->host, c->oscarHostEnt);
00495 }
```

```
00506 void connDoServerConnect(LPCONNECTION c)
00507 {
00508     connChangeState(c, CONN_STATE_CONNECT);
00509
00510     /* If there is no port, don't proceed.  The owner will
00511      * disconnect us.
00512      */
00513     if (c->serv->port == 0)
00514         return;
00515
00516     connConnectToHost(c, &c->ipaddr, c->serv->port);
00517 }
```

```
00569 void connDoValidation(LPCONNECTION c)
00570 {
00571     /* Only proceed if there is authorization.  If there is no
00572      * authorization, hang in this state until the owner explicitly
00573      * disconnects us.  This is used for auto-config.
00574      */
00575     if (c->auth != NULL)
00576     {
00577         /* Send the SIGNON packet.
00578         */
00579        connSendSignOn(c);
00580
00581        /* Nuke the auth structure.  It's no longer needed once the
00582         * SIGNON packet is sent.
00583        */
00584       if (c->auth)
00585       {
00586           MemFree(c->auth);
00587           c->auth = NULL;
00588       }
00589     }
00590
00591     connChangeState(c, CONN_STATE_VALIDATE);
00592 }


00599 void connEventLookupComplete(LPCONNECTION c)
00600 {
00601     int naddrs, index;
00602
00603     o_KillTimer(c->hSockWnd, TIMER_ID_LOOKUP);
00604     c->hLookupTask = 0;
00605
00606     switch (c->state)
00607     {
00608       case CONN_STATE_LOOKUP:
00609       {
00610          /* More than one address may be returned from the DNS
00611           * lookup.  Count how many there are and choose one at
00612           * random (for load distribution).
00613          */
00614         naddrs = 0;
00615         while (c->oscarHostEnt->h_addr_list[naddrs])
00616             naddrs++;
00617         index = naddrs == 1 ? 0 : (int)((GetTickCount()/1000) % naddrs);
00618
00619         /* Save these in case the attempt to connect to the first
00620          * host fails.  In this case, we will want to try other hosts.
00621         */
00622        c->numHosts = naddrs;
00623        c->initialHostIndex = c->currentHostIndex = index;
00624        _fmemcpy(&c->ipaddr,
00625               c->oscarHostEnt->h_addr_list[c->currentHostIndex],
00626              sizeof(IN_ADDR));
00627
00628       if (c->prox != NULL && c->prox->useProxy)
```
Page 10

```
00629                    connDoProxyLookup(c);
00630              else
00631                  connDoServerConnect(c);
00632              break;
00633          }
00634
00635      case CONN_STATE_PROXY_LOOKUP:
00636          {
00637              connDoProxyConnect(c);
00638              break;
00639          }
00640      }
00641 }
```

```
00739 void connEventAwaitChallangeComplete(LPCONNECTION c)
00740 {
00741     o_KillTimer(c->hSockwnd, TIMER_ID_CHALLANGE);
00742     connDoValidation(c);
00743 }
```

```
01048          case WM_GETHOSTBYNAME:
01049          {
01050              WORD error = WSAGETASYNCERROR(lParam);
01051              if (error == 0)
01052                  connEventLookupComplete(c);
01053              else
01054                  connEventLookupFailed(c);
01055              break;
01056          }
```

```
01063              if (event & FD_READ)
01064              {
01065                  if (c->state <= CONN_STATE_PROXY_REQUEST)
01066                  {
01067                      switch (c->prox->protocol)
01068                      {
01069                          case DLG_PROX_PROTO_SOCKS4:
```

```
                               conn.txt
01070                -              case DLG_PROX_PROTO_SOCKS5:
01071                                  connEventSocksResponse(c);
01072                .                 break;
01073                                 case DLG_PROX_PROTO_HTTPS:
01074             -                       connEventHttpsResponse(c);
01075                                     break;
01076                      }
01077            }
01078            else
01079                connEventRecvReady(c);
01080        }
```

```
01346 LPCONNECTION ConnCreate(LPVOID owner, CONNCALLBACK callback)
01347 {
01348     LPCONNECTION c;
01349
01350     if ((c = (LPCONNECTION)MemAlloc(sizeof(CONNECTION))) == NULL)
01351         return NULL;
01352
01353     if (!(c->hSockWnd = CreateWindow(CONN_CLASS, NULL, WS_POPUP,
01354                                     0, 0, 0, 0,
01355                                     NULL, NULL, lpOCMInfo->hModule, NULL)))
01356     {
01357         MemFree(c);
01358         return NULL;
01359     }
01360     SetWindowLong(c->hSockWnd, 0, (LONG)c);
01361
01362     c->callback         = callback;
01363
01364     if ((c->session = (LPSESSION)owner) != NULL)
01365         connCallInsertConnection(c, c->session);
01366
01367     c->state            = CONN_STATE_OFFLINE;
01368     c->sock             = INVALID_SOCKET;
01369     c->hLookupTask      = 0;
01370     c->oscarHostEnt     = NULL;
01371     c->proxyHostEnt     = NULL;
01372     c->serv             = NULL;
01373     c->prox             = NULL;
01374     c->auth             = NULL;
01375     c->proxResp         = NULL;
01376     c->destroyed        = FALSE;
01377     c->isReceiving      = FALSE;
01378     c->isBOSConnection  = FALSE;
01379     c->numHosts         = 0;
01380     c->initialHostIndex = 0;
01381     c->currentHostIndex = 0;
01382     c->numUnstartedSrvs = 0;
01383     c->numServices      = 0;
01384     c->serviceService   = NULL;
```

Page 22

```
01385        c->meterMap          = MapCreate();
01386        c->nInactivePeriods = 0;
01387        c->shutdownCount    = 0;
01388
01389        INIT_LIST(&c->services);
01390        INIT_LIST(&c->meters);
01391
01392        connInitSend(c);
01393        connInitRecv(c);
01394
01395        return c;
01396 }
```

```
01514 void ConnConnect(LPCONNECTION c, LPSERVCONFIG serv, LPPROXCONFIG prox,
01515                  LPAUTHINFO auth)
01516 {
01517     /* Reinitialize sending and receiving FSMs.
01518      */
01519     connInitSend(c);
01520     connInitRecv(c);
01521
01522     /* Save server info.
01523      */
01524     if (serv != NULL)
01525     {
01526         if (c->serv == NULL)
01527             c->serv = (LPSERVCONFIG)MemAlloc(sizeof(SERVCONFIG));
01528         _fmemcpy(c->serv, serv, sizeof(SERVCONFIG));
01529     }
01530
01531     /* Save proxy info.
01532      */
01533     if (prox != NULL)
01534     {
01535         if (c->prox == NULL)
01536             c->prox = (LPPROXCONFIG)MemAlloc(sizeof(PROXCONFIG));
01537         _fmemcpy(c->prox, prox, sizeof(PROXCONFIG));
01538     }
01539
01540     /* Save the authorization data until we are successfully connected.
01541      */
01542     if (auth != NULL)
01543     {
01544         if (c->auth == NULL)
01545             c->auth = (LPAUTHINFO)MemAlloc(sizeof(AUTHINFO));
01546         _fmemcpy(c->auth, auth, sizeof(AUTHINFO));
01547
01548         /* Assume it's the BOS connection if we have a username/password
01549          * authorization.  We need to set this immediately so that sess.c
01550          * doesn't screen out fatal errors during signon.
01551          */
01552         if (auth->type == AUTH_TYPE_USER)
01553             c->isBOSConnection = TRUE;
01554     }
01555     else
01556     {
01557         if (c->auth != NULL)
01558         {
01559             MemFree(c->auth);
01560             c->auth = NULL;
01561         }
01562     }
01563
01564     c->oscarHostEnt = (LPHOSTENT)MemAlloc(MAXGETHOSTSTRUCT);
01565
01566     /* If we're not using a proxy to do hostname resolution, initiate
01567      * a DNS lookup of the server.  Otherwise start the proxy connect
01568      * sequence by doing a DNS lookup of the proxy.
01569      */
01570     if (c->prox == NULL || !c->prox->useProxy ||
01571         (!c->prox->resolveHostnames &&
01572          (c->prox->protocol == DLG_PROX_PROTO_SOCKS4 ||
01573           c->prox->protocol == DLG_PROX_PROTO_SOCKS5)))
```
Page 25

```
01574          connDoServerLookup(c);
01575      else
01576          connDoProxyLookup(c);
01577 }
```

```
02159 void connReceiveBlock(LPCONNECTION c)
02160 {
02161     LPRECVSTATE r = &c->recvState;
02162     int n;
02163     BYTE buf[512];
02164     LPBYTE ptr;
02165
02166     if ((n = recv(c->sock, (char*)buf, 512, 0)) == SOCKET_ERROR)
02167         return;
02168     c->nInactivePeriods = 0;
02169
02170     ptr = buf;
02171
02172     while (n-- > 0)
02173     {
02174         BYTE b = *ptr++;
```

```
02237                    * buffer.
02238                    */
02239                   bytesToCopy = min(r->bytesLeft, (WORD)n);
02240                   _fmemcpy(r->currentByte, ptr, bytesToCopy);
02241
02242                   /* Adjust counts and pointers accordingly.
02243                    */
02244                   n -= bytesToCopy;
02245                   ptr += bytesToCopy;
02246                   r->bytesLeft -= bytesToCopy;
02247                   r->currentByte += bytesToCopy;
02248
02249                   /* If the packet is all here, dispatch it and begin
02250                    * waiting for a new packet.
02251                    */
02252                   if (r->bytesLeft == 0)
02253                   {
02254                       TRACERECVFLAP(inet_ntoa(c->ipaddr), r->type, r->seqNumber,
02255                                      r->length, r->data);
02256                       connProcessFLAP(c, r->type, r->length, r->data);
02257                       MemFree(r->data);
02258                       r->data = 0;
02259                       r->state = RECV_STATE_UNKNOWN;
02260                   }
02261                   break;
02262               }
02263           }
02264       }
02265 }


02268 void connEventRecvReady(LPCONNECTION c)
02269 {
02270     LPRECVSTATE r = &c->recvState;
02271
02272     /* Indicate that there's something to receive.
02273      */
02274     r->readyToReceive = TRUE;
02275
02276     /* Block reentry to the actual receiving code.  Reentry can happen
02277      * if a SNAC handler puts up a modal dialog box.
02278      */
02279     if (c->isReceiving)
02280         return;
02281     c->isReceiving = TRUE;
02282
02283     /* As long as data is available, read and process it.  Clear the
02284      * ready flag before processing, though, because it can be reset
02285      * during processing if a SNAC handler puts up a modal dialog.
02286      */
02287     while (r->readyToReceive)
02288     {
02289         r->readyToReceive = FALSE;
02290         connReceiveBlock(c);
02291
02292         /* If processing the received block caused the connection
02293          * to be destroyed, finish the job now and return immediately
02294          * without referencing the connection object again.
02295          */
02296         if (c->destroyed)
02297         {
02298             MemFree(c);
02299             return;
```

```
02300              }
02301         }
02302
02303         /* Allow entry again.
02304          */
02305         c->isReceiving = FALSE;
02306 }
```

```
02333 void connEventSendReady(LPCONNECTION c)
02334 {
02335         LPSENDSTATE s = &c->sendState;
02336         LPITEM item;
02337
02338         s->readyToSend = TRUE;
02339         while ((item = FIRST_ITEM(&s->queue)) != NULL_ITEM(&s->queue))
02340         {
02341             LPPACKET  packet = (LPPACKET)item;
02342             LPSERVICE service = PacketService(packet);
02343             LPQUEUE serviceQueue = PacketQueue(packet);
02344             WORD bytesAvailable;
02345             int bytesSent;  // must be signed to detect error return from send()
02346             LPBYTE addr;
02347
02348             /* Attempt to send as much of the block as possible.  Also
02349              * indicate that there has been activity on the connection.
02350              */
02351             PacketGetData(packet, &bytesAvailable, &addr);
02352             bytesSent = send(c->sock, addr, bytesAvailable, 0);
02353             c->nInactivePeriods = 0;
02354
02355             /* There is a chance that Winsock isn't really ready.  This
02356              * could happen if some other app snuck in and sent some data
02357              * before us.  It could also happen because of buggy Winsock
02358              * stacks (like the Shiva dialer from Netscape).  In any case,
02359              * the correct thing to do is to try again.
02360              */
02361             if (bytesSent == SOCKET_ERROR)
```

```
02362           {
02363               TraceMsg(TRACE_PROTO, "PROTO: Error sending data...retrying");
02364               return; -
02365           }
02366
02367           /* If only part of the packet could be sent, update its internal
02368            * pointers and return.  We will be called again when WinSock is
02369            * ready to accept more data.
02370            */
02371           if (bytesSent < (int)bytesAvailable)
02372           {
02373               PacketAdvance(packet, (WORD)bytesSent);
02374               s->readyToSend = FALSE;
02375               return;
02376           }
02377
02378           /* If the entire packet has been sent, destroy it.  This will
02379            * also remove it from its queue.  It's important to do this before
02380            * notifying the service, because the notification could cause
02381            * another packet to be queued, which could, in turn, cause this
02382            * to be reentered and send the old packet again.
02383            */
02384           PacketDestroy(packet);
02385
02386           /* Finally notivy the service.
02387            */
02388           if (service)
02389               ServPacketSent(service, serviceQueue);
02390       }
02391 }
02392
02393
02394 void ConnSendPacket(LPCONNECTION c, LPPACKET packet)
02395 {
02396       LPSENDSTATE s = &c->sendState;
02397
02398       PacketSetSequenceNumber(packet, s->seqNumber++);
02399
02400       TRACESENDFLAP(inet_ntoa(c->ipaddr),
02401                   PacketType(packet), PacketSequenceNumber(packet),
02402                   PacketContentSize(packet), PacketContentAddr(packet));
02403
02404       INSERT_ITEM_AT_TAIL((LPITEM)packet, &s->queue);
02405
02406       /* If the socket can accept data, attempt to send the block now.
02407        */
02408       if (s->readyToSend)
02409           connEventSendReady(c);
02410 }
```

```
02740 void SessSignOn(LPSESSION s, LPCSTR nickname, LPCSTR password)
02741 {
02742     LPCONNECTION connection;
02743     AUTHINFO authinfo;
02744
02745     /* Ignore this request if we are already signed on or are in the
02746      * process of signing on.
02747      */
02748     if (s->state > OM_PROTO_STATE_OFFLINE)
02749        return;
02750
02751     /* Create the initial connection.  It will initially be used
02752      * for authorization, but will ultimately become the first BOS
02753      * connection.
02754      */
02755     if ((connection = ConnCreate(s, sessConnCallback)) == NULL)
02756         return;
02757
02758     /* Establish the connection, using the supplied nickname and password
02759      * for authorization.
02760      */
02761     authinfo.type = AUTH_TYPE_USER;
02762     o_strncpy(authinfo.user.nickname, nickname, MAX_SZ_NICKNAME_LEN);
02763     o_strncpy(authinfo.user.password, password, MAX_SZ_PASSWORD_LEN);
02764
02765     ConnConnect(connection, &DlgServConfig, &DlgProxConfig, &authinfo);
02766
02767     /* Save the nickname so that other OCMs can enquire about it.  This
02768      * is temporary.  It will be replaced by the official nickname (with
02769      * capitalization and spacing from registration database) once we
02770      * retrieve that from the server.
02771      */
02772     SessSetNickname(s, nickname);
02773 }
```

```
03731 void connSendSignOn(LPCONNECTION c)
03732 {
03733     LPPACKET packet;
03734     BYTE verbuf[128], buf[512];
03735     WORD len;
03736     SNACSTREAM ss;
03737     LPAUTHINFO auth = c->auth;
03738
03739     SNACOpen(&ss, sizeof buf, buf);
03740
03741     SNACPutLong(&ss, 1);
03742
03743     switch (auth->type)
03744     {
03745       case AUTH_TYPE_USER:
03746       {
03747           WORD i;
03748           long temp;
03749           WORD nicksize = lstrlen(auth->user.nickname);
03750           WORD passsize = lstrlen(auth->user.password);
03751
03752           if (nicksize == 0 || passsize == 0)
03753           {
03754               ConnDisconnect(c);
03755               connCallFatalError(c, c->session, IDS_PASSWORD_REQUIRED, NULL);
03756               return;
03757           }
03758
03759           /* Send nickname and password.
03760            */
03761           SNACPutWord(&ss, TLV_TAGS_NICK);
03762           SNACPutWord(&ss, nicksize);
03763           SNACPutBlock(&ss, nicksize, (LPBYTE)auth->user.nickname);
03764
03765           SNACPutWord(&ss, TLV_TAGS_PASSWORD);
03766           SNACPutWord(&ss, passsize);
03767           for (i = 0; i < passsize; i++)
03768               SNACPutByte(&ss, (BYTE)(auth->user.password[i]^xorstring[i]));
03769
03770           /* Send client version string.
03771            */
03772           len = (WORD)wsprintf(verbuf, "%s, version %s/%s",
03773                   (LPCSTR)VERSION_PRODUCT_NAME,
03774                   (LPCSTR)VERSION_PRODUCT_VERSION_STRING,
03775                   VERSION_FILEOS == VOS_DOS_WINDOWS16 ?
03776                       (LPCSTR)"WIN16" : (LPCSTR)"WIN32");
03777           SNACPutWord(&ss, TLV_TAGS_CLIENT_IDENTITY);
03778           SNACPutWord(&ss, len);
03779           SNACPutBlock(&ss, len, verbuf);
03780
03781           /* Send structured client version.
03782            */
03783           {
03784               WORD maj, min, pnt, bld;
03785
03786               ParseVersionNumber(&maj, &min, &pnt, &bld);
03787
03788               SNACPutWord(&ss, TLV_TAGS_CLIENT_ID);
03789               SNACPutWord(&ss, 2);
03790               SNACPutWord(&ss, TOOL_ID);
03791
03792               SNACPutWord(&ss, TLV_TAGS_MAJOR_VERSION);
```

```
03793                SNACPutWord(&ss, 2);
03794                SNACPutWord(&ss, maj);
03795
03796                SNACPutWord(&ss, TLV_TAGS_MINOR_VERSION);
03797                SNACPutWord(&ss, 2);
03798                SNACPutWord(&ss, min);
03799
03800                SNACPutWord(&ss, TLV_TAGS_POINT_VERSION);
03801                SNACPutWord(&ss, 2);
03802                SNACPutWord(&ss, pnt);
03803
03804                SNACPutWord(&ss, TLV_TAGS_BUILD_NUM);
03805                SNACPutWord(&ss, 2);
03806                SNACPutWord(&ss, bld);
03807            }
03808
03809            /* Send the international crap.
03810             */
03811            {
03812                char buf[32];
03813
03814                if (RDBLoadString(RESMODULE, IDS_INTL_COUNTRY_CODE,
03815                                  buf, sizeof buf))
03816                {
03817                    SNACPutWord(&ss, TLV_TAGS_COUNTRY);
03818                    SNACPutString(&ss, buf);
03819                }
03820                if (RDBLoadString(RESMODULE, IDS_INTL_LANGUAGE_CODE,
03821                                  buf, sizeof buf))
03822                {
03823                    SNACPutWord(&ss, TLV_TAGS_LANGUAGE);
03824                    SNACPutString(&ss, buf);
03825                }
03826                if (RDBLoadString(RESMODULE, IDS_INTL_SCRIPT_CODE,
03827                                  buf, sizeof buf))
03828                {
03829                    SNACPutWord(&ss, TLV_TAGS_SCRIPT);
03830                    SNACPutString(&ss, buf);
03831                }
03832            }
03833
03834            /* If it's a non-AOL client, send the distribution channel
03835             */
03836            temp = RDBLoadValue(RESMODULE, IDV_DIST_CHANNEL, 0);
03837            if (temp != 0)
03838            {
03839                SNACPutWord(&ss, TLV_TAGS_DIST_CHANNEL);
03840                SNACPutWord(&ss, 4);
03841                SNACPutLong(&ss, (DWORD)temp);
03842            }
03843
03844            /* If there is a stored disconnect reason, send it now.
03845             */
03846            temp = ProfGetLong(PROF_GLOBAL, MISC_GROUP, MISC_KEY_DISCONNECT);
03847            if (temp != 0)
03848            {
03849                SNACPutWord(&ss, TLV_TAGS_DISCONNECT_REASON);
03850                SNACPutWord(&ss, 2);
03851                SNACPutWord(&ss, (WORD)temp);
03852            }
03853
03854            /* Mark this connection as the BOS connection.
03855             */
```

```
03856              c->isBOSConnection = TRUE;
03857
03858          break;          -
03859        }
03860      case AUTH_TYPE_COOKIE:
03861        {
03862          SNACPutWord(&ss, TLV_TAGS_LOGIN_COOKIE);
03863          SNACPutWord(&ss, auth->cookie.length);
03864          SNACPutBlock(&ss, auth->cookie.length, auth->cookie.buffer);
03865          break;
03866        }
03867      }
03868
03869
03870      packet = PacketCreate(FLAP_SIGNON_TYPE, SNACBytesTransferred(&ss), buf,
03871                            NULL, NULL);
03872      ConnSendPacket(c, packet);
03873 }
```

```
04052 void connProcessSignOn(LPCONNECTION c, LPSNACSTREAM lpss)
04053 {
04054     DWORD version;
04055     SNACGetLong(lpss, &version);
04056     if (version == 1)
04057         connEventAwaitChallangeComplete(c);
04058     else
04059         connEventAwaitChallangeFailed(c);
04060 }
```

```
04628 void connProcessFLAP(LPCONNECTION c, WORD type, WORD size, LPBYTE data)
04629 {
04630     SNACSTREAM ss;
04631
04632     SNACOpen(&ss, size, data);
04633
04634     switch (type)
04635     {
04636       case FLAP_SIGNON_TYPE:
04637           connProcessSignOn(c, &ss);
04638           break;
04639
04640       case FLAP_SIGNOFF_TYPE:
04641           connProcessSignOff(c, &ss);
04642           break;
04643
04644       case FLAP_ERROR_TYPE:
04645           connProcessError(c, &ss);
04646           break;
04647
04648       case FLAP_DATA_TYPE:
04649       {
04650           WORD  group, type, flags;
04651           DWORD reqid;
04652           SNACGetHeader(&ss, &group, &type, &flags, &reqid);
04653           if (group == OMGROUP_SERVICE)
04654           {
04655               switch (type)
04656               {
04657                 case SNAC_SNAC_ERR:
04658                 {
04659                     WORD code, type, len;
04660
04661                     SNACGetWord(&ss, &code);
04662                     while (SNACGetWord(&ss, &type))
04663                     {
04664                         SNACGetWord(&ss, &len);
04665                         SNACSkipBytes(&ss, len);
04666                     }
04667
04668                     switch (GETTAG(reqid))
04669                     {
04670                       case SNAC_SERVICE_CLIENT_ONLINE:
```
Page 75

```
04734                    break;
04735                case SNAC_SERVICE_MOTD:
04736                    connProcessMOTD(c, &ss, reqid);
04737                    break;
04738            }
04739        }
04740        else
04741        {
04742            OMSendMessage(OMTYPE_SNAC, group, type, size, data);
04743        }
04744        break;
04745    }
04746  }
04747 }
```

OscaR — □ ✕

File   People

Favorites

AOL
Instant
Messenger

Online | List Setup |

▼ **Buddies (1/1)**
OscaRaina
▼ **Family (0/0)**
▼ **Co-Workers (0/0)**

Search the Web

Try **AOL** *Today!*

File Transfer ✕

| Virus Checker | | Away | | Buddy Chat |

| Sounds | General | Connection | Controls | Buddy | **File Transfer** |

When others issue the File Send command:
- ○ Block all incoming invitations
- ◉ Always display the Receive File Dialog
- ○ Automatically accept files from users on my Buddy List
- ○ Automatically accept files from everyone

When others issue the File Get command:
- ○ Allow no users to get my files
- ◉ Allow only users on my Buddy List to get my files
- ○ Allow everyone to get my files
- ☐ Never display Status dialog
- ☐ Keep a record in logfile.txt of who has gotten files

Directory from where others can get my files or I send files:

C:\temp\

Directory to put files I receive:

C:\WINNT\aim95\oscarlogan\download\

Initial transfer speed:
To Buddy:
- ◉ Fast
- ○ Medium
- ○ Slow

From Buddy:
- ◉ Fast
- ○ Medium
- ○ Slow

Port number to use:

[OK]   [Cancel]   [Apply]

Exhibit 4

Exhibit 5

Online | List Setup

- ▼ **Buddies (1/2)**
  - OscaRaina
- ▼ **Family (0/0)**
- ▼ **Co-Workers (0/0)**
- ▼ Other (1/2)

Search the Web

7.9 +54   INDEX:INX 114

Prices delayed at least 15 minute:

OscaRaina: Yes, you can get it by using the GetFile menu command
oscaremily wants to get list of files.

This is the list of files you can get from your buddy. Select the file you want and click the 'Get' button. Hold down the Ctrl key while using the mouse button to select several files.

02/20/1999	11:19	2505	oscaraina\nesh.gif
02/20/1999	11:20	2344	oscaraina\endlesstriangle.gif
02/20/1999	11:21	790	oscaraina\profileblk.gif
02/20/1999	11:21	790	oscaraina\profilewht.gif
02/20/1999	11:22	2594	oscaraina\circlet.gif
02/20/1999	11:23	1645	oscaraina\gridopp.gif
10/10/2000	14:25	2153	oscaraina\liar.gif
02/20/1999	11:23	2528	oscaraina\lift.gif
02/20/1999	11:24	1919	oscaraina\parallel.gif
02/20/1999	11:25	2283	oscaraina\sax.gif
02/20/1999	11:25	1219	oscaraina\horizontal.gif
01/02/1993	00:46	2707	oscaraina\emacs.xbm
02/20/1999	11:20	1309	oscaraina\bar\great.gif

☑ Check file for virus after transfer   ☐ Open file after transfer

[Get]   [Stop]   [Cancel]

Exhibit 6

osca... _ □ X

File People
Favorites

AOL
Instant
Messenger

Online | List Setup

▼ **Buddies (1/1)**
  OscaRaina
▼ **Family (0/0)**
▼ **Co-Workers (0/0)**

Search the Web

try AOL Today!

OscaRaina - Instant Mes _ □ X

File Edit View People

oscarlogan: Do you have the proposal done?
OscaRaina: Yes, you can get it by using the GetFile
menu command.

Get File from OscaRaina X

WARNING! You are about to retrieve a file. Be cautious,
any files you retrieve may contain a computer virus. Computer
viruses can destroy your work and even damage your computer.
AOL Instant Messenger (SM) Anti-virus Center:

How can I protect my computer?

1. Get anti-virus software, and use it.
2. Don't open file transfers from strangers.
3. Update your anti-virus software regularly.
4. Make back-ups of your important files regularly.
5. Run your own virus-check on any file someone sends you.
6. If you exchange floppy disks, virus-check your floppies.

Some Leading Anti-virus Software:

Norton AntiVirus V5.0 for Windows 95/98
VirusScan Classic V4.0
Dr. Solomon's Anti-Virus Deluxe Windows 95/NT

☐ Don't ask me again! [OK] [Cancel]

Exhibit 7

Exhibit 8.

OSC — □ X

File People
Favorites

AOL
Instant
Messenger

Online | List Setup

▼ Buddies (1/1)
    OscaRaina
▼ Family (0/0)
▼ Co-Workers (0/0)

Search the Web   ▶ Go

Try AOL Today!

Get File request fro X

OscaRaina
wants to get files from your disk directory
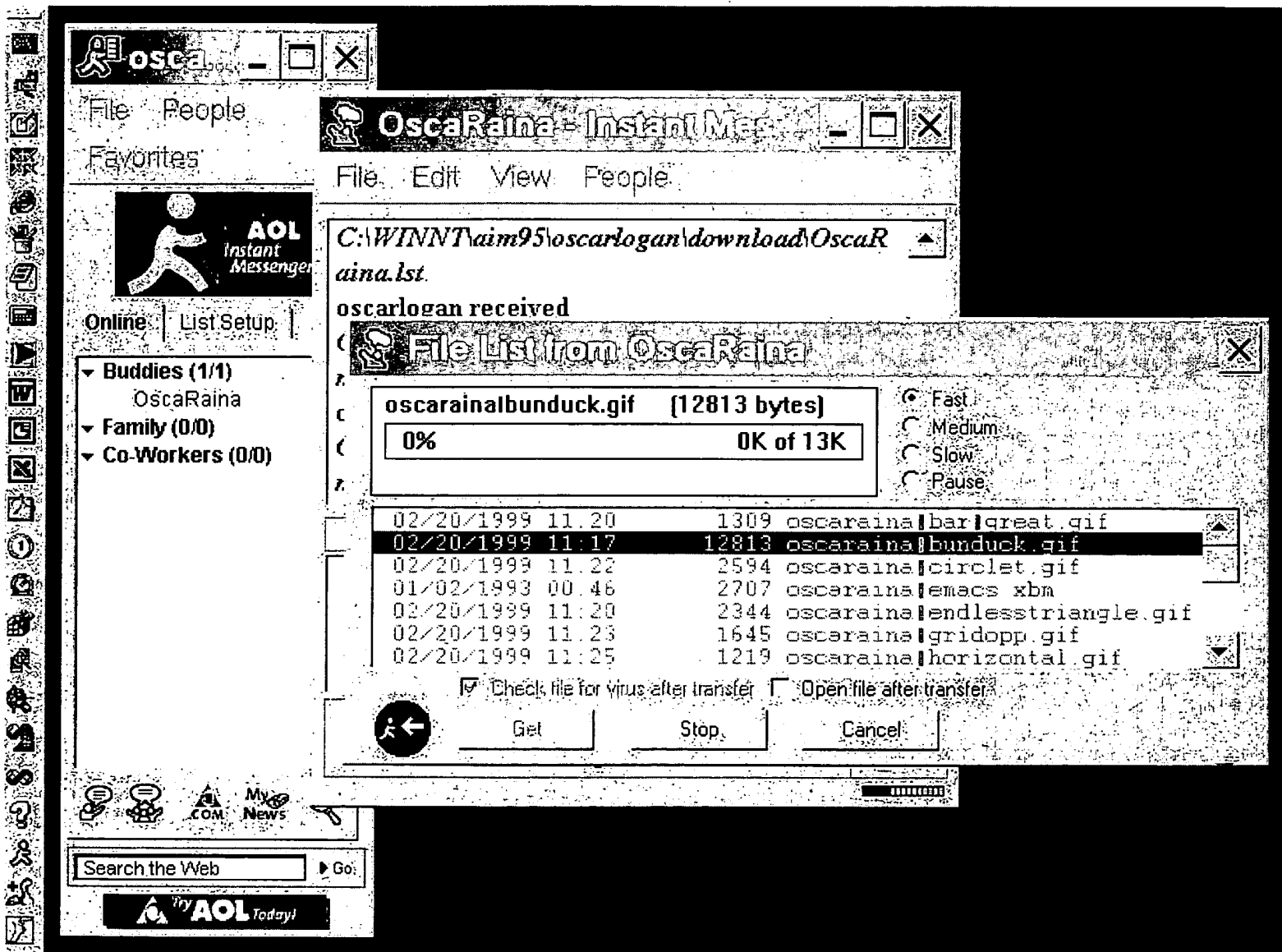C:\WINDOWS\aim95\oscarlogan\filelib\
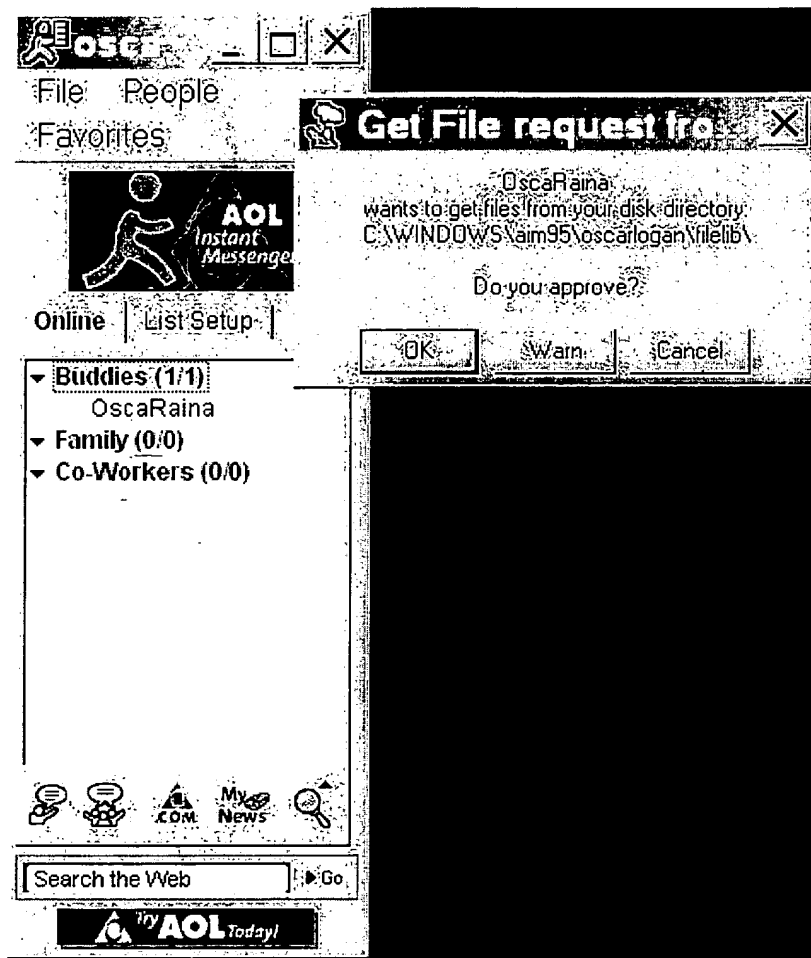
Do you approve?

OK | Warn | Cancel

Exhibit 9

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

☑ **BLACK BORDERS**

☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**

☐ **FADED TEXT OR DRAWING**

☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**

☐ **SKEWED/SLANTED IMAGES**

☑ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**

☐ **GRAY SCALE DOCUMENTS**

☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**

☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**

☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.